

AIX 6 Basics

(Course Code AU13)

Instructor Guide

ERC 10.0

IBM Certified Course Material

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM® is a registered trademark of International Business Machines Corporation.

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX®	AIX 5L™	Common User Access®
MVS™	OS/2®	pSeries®
System p™	System p5™	400®

PS/2® is a trademark or registered trademark of Lenovo in the United States, other countries, or both.

Adobe is either a registered trademark or a trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX® is a registered trademark of The Open Group in the United States and other countries.

Linux® is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

February 2008 Edition

The information contained in this document has not been submitted to any formal IBM test and is distributed on an "as is" basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will result elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

© Copyright International Business Machines Corporation 1995, 2008. All rights reserved.

This document may not be reproduced in whole or in part without the prior written permission of IBM.

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Trademarks	xi
Instructor Course Overview	xiii
Course Description	xv
Agenda	xvii
Unit 1. Introduction to AIX	1-1
Unit Objectives	1-2
AIX Operating System	1-4
Working on an AIX System (1 of 2)	1-6
Working on an AIX System (2 of 2)	1-9
Activity: Fill in the Blanks	1-12
The Shell: User Interface to AIX	1-14
Useful AIX Utilities	1-17
AIX Graphical User Interfaces	1-19
Checkpoint	1-22
Unit Summary	1-24
Unit 2. Using the System	2-1
Unit Objectives	2-2
Logging In and Out	2-4
Passwords	2-7
Command Format	2-9
Command Format Examples	2-12
The <code>date</code> and <code>cal</code> Commands	2-14
The <code>clear</code> , <code>echo</code> , and <code>banner</code> Commands	2-16
Activity: Questions and Answers	2-18
The <code>who</code> and <code>finger</code> Commands	2-20
Sending Mail	2-23
Receiving Mail	2-26
The <code>write</code> and <code>wall</code> Commands	2-29
<code>talk</code> with Another User	2-32
<code>mesg</code>	2-34
Keyboard Tips	2-36
Checkpoint	2-38
Exercise: Using The System	2-40
Unit Summary	2-42
Unit 3. AIX 6.1 Documentation	3-1
Unit Objectives	3-2
<code>man</code> Command	3-4
<code>man</code> Example	3-6

<i>man -k</i> : Working with a Keyword	3-8
Viewing AIX 6.1 Documentation	3-10
Accessing the Documents from a Web Browser	3-12
AIX 6.1 Documentation	3-14
Search AIX 6.1 Documentation	3-17
Search Scope	3-19
Checkpoint	3-21
Exercise: AIX 6.1 Documentation	3-23
Unit Summary	3-25
Unit 4. Files and Directories	4-1
Unit Objectives	4-2
A File	4-4
File Types	4-6
Directory Contents	4-9
AIX File Systems	4-12
Hierarchical Structure	4-14
Path Names	4-18
Where Am I?	4-20
Listing Directories	4-22
Long Listing of Files	4-24
Change Current Directory	4-28
Activity: Q + A	4-30
Creating Directories	4-32
Removing Directories	4-34
Working with Multiple Directories	4-36
Displaying Directory Information	4-38
AIX File Names	4-40
<i>touch</i> Command	4-42
Checkpoint (1 of 2)	4-44
Checkpoint (2 of 2)	4-46
Exercise: Files and Directories	4-48
Unit Summary	4-50
Unit 5. Using Files	5-1
Unit Objectives	5-2
Copying Files	5-4
cp Examples	5-6
Moving and Renaming Files	5-9
mv Examples	5-11
Listing File Contents	5-13
Displaying Files	5-15
<i>wc</i> Command	5-17
Activity: Working with the wc Command	5-19
Linking Files	5-22
Linking Files (cont.)	5-24
Removing Files	5-26
Printing Files	5-28

Checkpoint	5-31
Exercise: Using Files	5-33
Unit Summary	5-35
Unit 6. File Permissions	6-1
Unit Objectives	6-2
Long Listing of Files	6-4
File Protection/Permissions	6-7
Changing Permissions (Symbolic Notation)	6-9
Changing Permissions (Octal Notation)	6-12
Default File Permissions	6-15
<u>umask</u>	6-17
Activity: Personal Directories	6-20
Function/Permissions Required	6-23
Checkpoint (1 of 3)	6-25
Checkpoint (2 of 3)	6-27
Checkpoint (3 of 3)	6-29
Exercise: File Permissions	6-31
Unit Summary	6-33
Unit 7. The vi Editor	7-1
Unit Objectives	7-2
Introduction to the vi Editor	7-4
Starting vi	7-7
Adding Text	7-9
Exiting the Editor	7-11
Cursor Movement	7-13
Deleting Text	7-16
Search for a Pattern	7-18
Activity: vi Commands	7-20
Changing Text	7-22
Cut, Copy, and Paste	7-25
vi - Executing AIX Commands	7-28
vi Options	7-31
Command Line Editing	7-34
vi Editors	7-37
Checkpoint	7-39
Exercise: vi Editor	7-41
Unit Summary	7-43
Unit 8. Shell Basics	8-1
Unit Objectives	8-2
The Shell	8-4
Metacharacters and Wildcards	8-6
File Name Substitution (1 of 2)	8-8
File Name Substitution (2 of 2)	8-10
The Standard Files	8-13
File Descriptors	8-15

Input Redirection	8-17
Output Redirection	8-19
Creating a File with <code>cat</code>	8-21
Activity: Review Shell Basics	8-23
Error Redirection	8-25
Combined Redirection	8-28
Pipes	8-31
Filters	8-33
Split Outputs	8-35
Command Grouping	8-37
Line Continuation	8-39
Checkpoint (1 of 2)	8-41
Checkpoint (2 of 2)	8-43
Exercise: Shell Basics	8-45
Unit Summary	8-47
Unit 9. Using Shell Variables	9-1
Unit Objectives	9-2
Shell Variables	9-4
Listing Variable Settings	9-6
Setting and Referencing Shell Variables	9-8
Shell Variables Example	9-10
Command Substitution	9-12
Quoting Metacharacters	9-14
Command Line Parsing	9-16
Checkpoint (1 of 2)	9-18
Checkpoint (2 of 2)	9-20
Exercise: Using Shell Variables	9-22
Unit Summary	9-24
Unit 10. Processes	10-1
Unit Objectives	10-2
What Is a Process?	10-4
Login Process Environment	10-6
Process Environment	10-8
Parents and Children	10-11
Variables and Processes	10-14
Activity: Exporting Variables	10-16
What Is a Shell Script?	10-21
Invoking Shell Scripts (1 of 3)	10-23
Invoking Shell Scripts (2 of 3)	10-25
Invoking Shell Scripts (3 of 3)	10-27
Exit Codes from Commands	10-29
Checkpoint	10-31
Activity: Shell Scripts	10-33
Unit Summary	10-36

Unit 11. Controlling Processes	11-1
Unit Objectives	11-2
Monitoring Processes	11-4
Controlling Processes	11-6
Terminating Processes (1 of 2)	11-9
Terminating Processes (2 of 2)	11-11
Signals	11-14
Running Long Processes	11-17
Job Control in the Korn Shell	11-20
Job Control Example	11-23
Daemons	11-25
Checkpoint	11-27
Exercise: Controlling Processes	11-29
Unit Summary	11-31
Unit 12. Customizing the User Environment.....	12-1
Unit Objectives	12-2
Login Files	12-4
Sample <i>/etc/environment</i>	12-7
Sample <i>/etc/profile</i>	12-9
Environment Variables (1 of 2)	12-11
Sample <i>.profile</i>	12-13
Environment Variables (2 of 2)	12-15
Sample <i>.kshrc</i>	12-18
<i>ksh</i> Features - Aliases	12-20
<i>ksh</i> Features - Using Aliases	12-23
<i>ksh</i> Features - History	12-25
Checkpoint	12-27
Exercise: Customizing the User Environment	12-29
Unit Summary	12-31
Unit 13. AIX Utilities, Part I.....	13-1
Unit Objectives	13-2
<i>find</i>	13-4
Sample Directory Structure	13-7
Using <i>find</i>	13-9
Executing Commands with <i>find</i>	13-11
Interactive Command Execution	13-13
Additional Options	13-15
The Shell versus <i>find</i>	13-17
<i>find</i> Examples	13-19
AIX Utilities (1)	13-21
<i>grep</i>	13-23
<i>grep</i> Sample Data Files	13-25
Basic <i>grep</i>	13-27
<i>grep</i> with Regular Expressions	13-29
<i>grep</i> Examples	13-32
<i>grep</i> Options	13-34

Other grep Commands	13-36
Activity: grep Command	13-39
sort Command	13-42
sort Examples	13-44
head and tail Commands	13-46
telnet: Login to Remote Hosts	13-49
ftp : Transfers Files Between Hosts	13-51
ftp Subcommands	13-53
rexec, rsh: Non-interactive Remote Execution	13-55
Secure Shell Utilities (OpenSSH)	13-58
tar : Backup and Restore Files	13-61
Checkpoint	13-63
Exercise: AIX Utilities (2)	13-65
Unit Summary	13-67
 Unit 14. AIX Utilities, Part II	 14-1
Unit Objectives	14-2
xargs	14-4
xargs Examples	14-7
xargs , find , and grep	14-9
The -links Option with find	14-12
alias and find	14-14
which , whereis , and whence	14-16
file	14-19
Exercise: AIX Utilities (3)	14-22
diff (Differential File Comparator)	14-24
Comparing Two Files Using diff	14-26
Comparing Two Files Using cmp	14-29
Comparing Directories Using dirdiff	14-31
compress , uncompress , and zcat	14-34
Displaying Non-Printable Characters in Files	14-37
Non-Printable Characters in Directories	14-39
Assigning Unique File Names	14-41
Checkpoint	14-43
Exercise: AIX Utilities (4)	14-45
Unit Summary	14-47
 Unit 15. Additional Shell Features	 15-1
Unit Objectives	15-2
Important Shell Variables	15-4
Positional Parameters	15-6
The expr Utility	15-8
expr Examples	15-11
Conditional Execution	15-13
test Command	15-15
if Command	15-17
Activity: Writing Shell Scripts	15-20
read Command	15-23

for Loop Syntax	15-25
while Loop Syntax	15-27
Command Search Order	15-29
Sample .profile	15-31
Checkpoint	15-34
Exercise: Additional Shell Features	15-36
Unit Summary	15-38
Unit 16. The AIX Graphical User Interface	16-1
Unit Objectives	16-2
The X Window System	16-4
What is AIXwindows?	16-7
An X Window Network Configuration	16-9
The Client/Server Environment	16-12
X Clients	16-14
The X Server	16-17
Starting AIXwindows	16-19
Stopping X	16-22
An AIXwindows Display	16-24
The aixterm Window	16-26
Running a Client on Another System	16-29
The xhost Command	16-32
The xauth Command	16-35
Common Desktop Environment (CDE)	16-38
The Components of the CDE Desktop	16-41
The Login Manager	16-43
\$HOME/.dtprofile	16-45
Front Panel	16-48
Front Panel - Subpanels	16-51
Front Panel - Further Controls	16-54
The Style Manager	16-56
The File Manager	16-58
The Application Manager	16-60
The Personal Applications Manager	16-62
The Terminal Emulator	16-64
The Help System	16-66
Checkpoint Questions (1 of 2)	16-68
Checkpoint Questions (2 of 2)	16-70
Exercise: Using AIXwindows and CDE	16-72
Unit Summary	16-74

Appendix A. Checkpoint Solutions	A-1
Appendix B. Command Summary	B-1
Appendix C. Customizing AIXwindows.....	C-1
Appendix D. CDE User Customization	D-1
Glossary	X-1

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM® is a registered trademark of International Business Machines Corporation.

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX®	AIX 5L™	Common User Access®
MVS™	OS/2®	pSeries®
System p™	System p5™	400®

PS/2® is a trademark or registered trademark of Lenovo in the United States, other countries, or both.

Adobe is either a registered trademark or a trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX® is a registered trademark of The Open Group in the United States and other countries.

Linux® is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Instructor Course Overview

This course has been designed from a user's perspective. The students are not required to have any prior knowledge of AIX or any other UNIX-based system.

The course sessions have been designed in a logical order to enable the novice user to identify the major components of AIX. Then, the students are introduced to the operating system by logging in and out of the system and carrying out a few basic operations. The hierarchical tree structure is explained in detail, and all the functions that can be carried out on files and directories. The concept of a shell is introduced and the operations that are supported through it. The one editor that is covered is **vi**, this being the one that is available on most UNIX platforms. The concept of users owning jobs and thus processes is introduced, including the environment in which processes execute. Finally, to pull together all the ideas from the previous units, a few useful tools are introduced which help users customize their environments and write very simple shell scripts, which is the main objective of the course.

The course objectives and content are listed in this section. Ensure that you cover both at the start of the course to set the correct expectations right from the start.

Also note that all your students will have different levels of experience with the operating system and also different capacities in learning. So, it is very important to identify the level of your audience at the beginning and only introduce additional information/discussion items where relevant. The discussion items provided can usually be used to challenge the more experienced students.

The main teaching aid that is provided with the course is the course visuals. However, feel free to demonstrate any of the concepts covered on demonstration workstations or in any other way, if you feel that by doing so you will aid the learning process of the students.

Finally, remember that this course has been designed for users of the operating system. System administrative concepts should not be covered because they are outside the scope of this course. For students who are interested in administrative tasks, encourage them to attend the System Administration and the Advanced System Administration courses. The System Administration course covers how a system can be set up from a new installation, whereas the emphasis for the advanced course is more on problem determination.

Course Description

AIX 6 Basics

Duration: 4 days

Purpose

This course enables students to perform everyday tasks using the AIX operating system.

Audience

This course is suitable for anyone who requires basic AIX user skills. This course is also a prerequisite for students who plan to attend the AIX System Administration courses.

Prerequisites

Students attending this course should be familiar with basic information technology (IT) concepts and the role of an operating system.

Objectives

After completing this course, you should be able to:

- Log in to an AIX system and set a user password
- Use AIX online documentation
- Manage AIX files and directories
- Describe the purpose of the shell
- Use the vi editor
- Execute common AIX commands and manage AIX processes
- Customize the working environment
- Use common AIX utilities
- Write simple shell scripts
- Use the AIXWindows environment
- Use the Common Desktop Environment

Contents

- Introduction to AIX
- Using the System
- AIX 6.1 Documentation
- Files and Directories
- Using Files
- File Permissions
- The vi Editor
- Shell Basics
- Using Shell Variables
- Processes
- Customizing the User Environment
- AIX Utilities, Part I
- AIX Utilities, Part II
- Additional Shell Features
- The AIX Graphical User Interface

Curriculum relationship

This course is the first course in the AIX curriculum and is a prerequisite for all the training paths.

Agenda

Day 1

- (00:20) Welcome
- (00:30) Unit 1 - Introduction to AIX
- (00:45) Unit 2 - Using the System
- (00:30) Exercise 1 - Using the System
- (00:45) Unit 3 - AIX Documentation
- (00:45) Exercise 2 - AIX Documentation
- (01:15) Unit 4 - Files and Directories
- (00:30) Exercise 3 - Files and Directories
- (00:40) Unit 5 - Using Files
- (00:45) Exercise 4 - Using Files

Day 2

- (01:00) Unit 6 - File Permissions
- (00:45) Exercise 5 - File Permissions
- (00:45) Unit 7 - The vi Editor
- (00:45) Exercise 6 - The vi Editor
- (01:00) Unit 8 - Shell Basics
- (00:45) Exercise 7 - Shell Basics
- (00:40) Unit 9 - Using Shell Variables
- (00:45) Exercise 8 - Using Shell Variables

Day 3

- (00:50) Unit 10 - Processes
- (00:45) Unit 11 - Controlling Processes
- (00:45) Exercise 9 - Controlling Processes
- (00:30) Unit 12 - Customizing the User Environment
- (00:30) Exercise 10 - Customizing the User Environment
- (00:25) Unit 13 - AIX Utilities, Part I
- (00:30) Exercise 11 - AIX Utilities (1)
- (00:30) Unit 13 - AIX Utilities, Part I (Continued)
- (00:45) Exercise 12 - AIX Utilities (2)
- (00:25) Unit 14 - AIX Utilities, Part II
- (00:30) Exercise 13 - AIX Utilities (3)

Day 4

- (00:25) Unit 14 - AIX Utilities, Part II (Continued)
- (00:35) Exercise 14 - AIX Utilities (4)
- (00:45) Unit 15 - Additional Shell Features
- (01:00) Exercise 15 - Additional Shell Features
- (01:30) Unit 16 - The AIX Graphical User Interface

Text highlighting

The following text highlighting conventions are used throughout this book:

Bold	Identifies file names, file paths, directories, user names, and principals.
<i>Italics</i>	Identifies links to Web sites, publication titles, and is used where the word or phrase is meant to stand out from the surrounding text.
Monospace	Identifies attributes, variables, file listings, SMIT menus, code examples of text similar to what you might see displayed, examples of portions of program code similar to what you might write as a programmer, and messages from the system.
Monospace bold	Identifies commands, daemons, menu paths, and what the user would enter in examples of commands and SMIT menus.
<text>	The text between the < and > symbols identifies information the user must supply. The text may be normal highlighting, bold or monospace, or monospace bold depending on the context.

Unit 1. Introduction to AIX

Estimated Time

00:30

What This Unit Is About

This unit is an introduction to the course AIX 6 Basics.

What You Should Be Able to Do

After completing this unit, students should be able to:

- Describe the major components of an AIX system
- Describe the major topics in this course
- Explain the value of these topics when working in an AIX environment

How You Will Check Your Progress

Accountability:

- Student activity
- Checkpoint questions

Unit Objectives

After completing this unit, you should be able to:

- Describe the major components of an AIX system
- Describe the major topics in this course
- Provide the value of these topics when working in an AIX environment

© Copyright IBM Corporation 2008

Figure 1-1. Unit Objectives

AU1310.0

Notes:

Instructor Notes:

Purpose — Explain what students will learn in this unit.

Details —

Additional Information — None.

Transition Statement — Let's start with a high-level view of the AIX operating system.

AIX Operating System

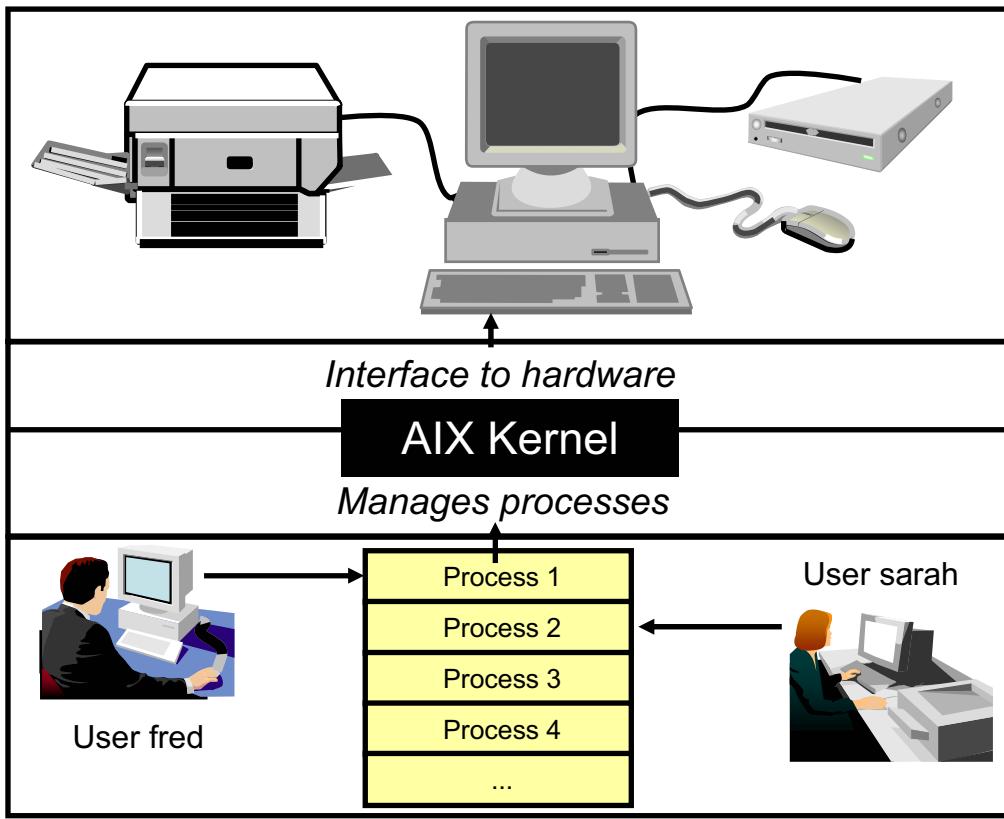


Figure 1-2. AIX Operating System

AU1310.0

Notes:

The AIX Kernel

A computer consists of many hardware devices that the users of a computer system want to use. For example, they want to print documents or they want to play a game from a CD-ROM.

To control these hardware devices and to share them between multiple users, an operating system must be loaded during the system startup. In the case of the AIX operating system, there is one special program which interfaces directly to the hardware devices: *the AIX Kernel*. The Kernel controls the access to the devices.

On the other hand, the users start different programs, for example, a program that prints a document or removes a file. These programs that run in AIX processes are also controlled by the AIX Kernel.

To say it simply: The AIX Kernel is the heart of your operating system.

Instructor Notes:

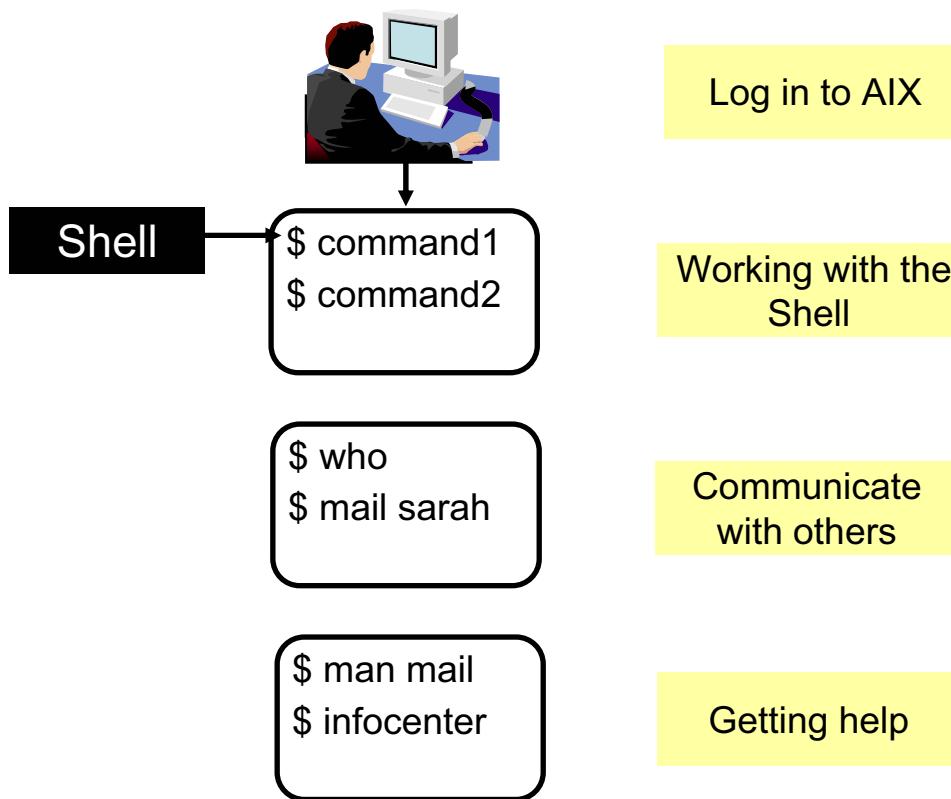
Purpose — Describe the role of an operating system and the AIX Kernel.

Details — Give an easy introduction. Keep things simple and explain as described in the notes.

Additional Information —

Transition Statement — Let's introduce what students will learn in this course.

Working on an AIX System (1 of 2)



© Copyright IBM Corporation 2008

Figure 1-3. Working on an AIX System (1 of 2)

AU1310.0

Notes:

Log in

AIX is a *multi-user system*. Before a user can work with AIX, an authentication process takes place. The user must log in with the user's username and password.

The shell

After a successful authentication, AIX starts a certain program for the user, a *shell*. The shell is a *command interpreter* that waits for input and executes the commands and programs the user types in. As you will learn in this course, the shell is not only a command interpreter; it offers great flexibility. Working with the shell is one of the major topics in this course.

Communication

Multiple users can work at the same time on an AIX system or in a network. One of the basic tasks in your daily work is to communicate with other users on a system or in the network. In this course, you will learn different commands that allow communication with other users.

Additional information

AIX offers a wide range of tools and commands. There are multiple ways to obtain assistance with commands; for example, the `man` command or the *AIX Online Documentation*. How to work with these help tools is also a major topic in this course.

Instructor Notes:

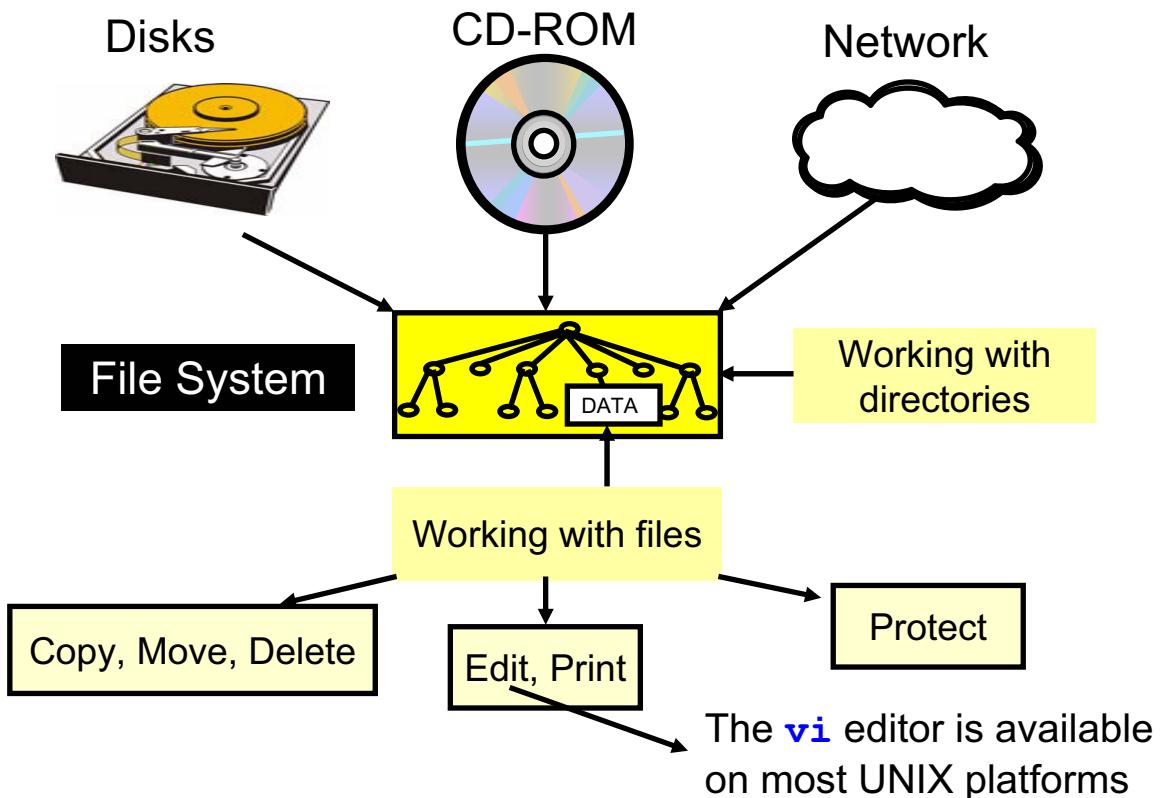
Purpose — Provide the major topics in the course by explaining this visual.

Details —

Additional Information —

Transition Statement — Let's look at some more areas that will be covered in this course.

Working on an AIX System (2 of 2)



© Copyright IBM Corporation 2008

Figure 1-4. Working on an AIX System (2 of 2)

AU1310.0

Notes:

AIX file structure

One of the major tasks of a computer system is to read and write data. In order to do this, AIX uses a *hierarchical file tree* that consists of directories, subdirectories, and files. The *top level directory* is called the root (/) directory that has many subdirectories. Each of these subdirectories can contain files or other subdirectories. A directory is like a folder in which you put certain documents.

File system types

The file tree is mounted during the system startup. AIX supports different file system types, which are all mounted to one big file tree. This is shown on the visual. Parts of this file tree reside on a disk, other parts may reside on a CD-ROM, or are mounted from another computer in a network.

What you will learn

This course explains how to work with directories and files on a user level. You will learn how to navigate in the file tree and how to manage directories. You will learn how to copy, move, delete and print files, and how to edit files using **vi**, which is the common UNIX editor. Another topic will show how to specify correct file permissions.

Instructor Notes:

Purpose — Explain the terms file systems, file tree, directory, and file and how these components can be found in the course.

Details — Mention the `vi` editor because there is one question in the checkpoint.

Also, point out that the `vi` editor is very important in this course and there will be an entire unit on `vi` later in the course.

We will have a closer look at directories (file systems) later in this course.

Additional Information —

Transition Statement — Let's have a short activity to review some terms.

Activity: Fill in the Blanks

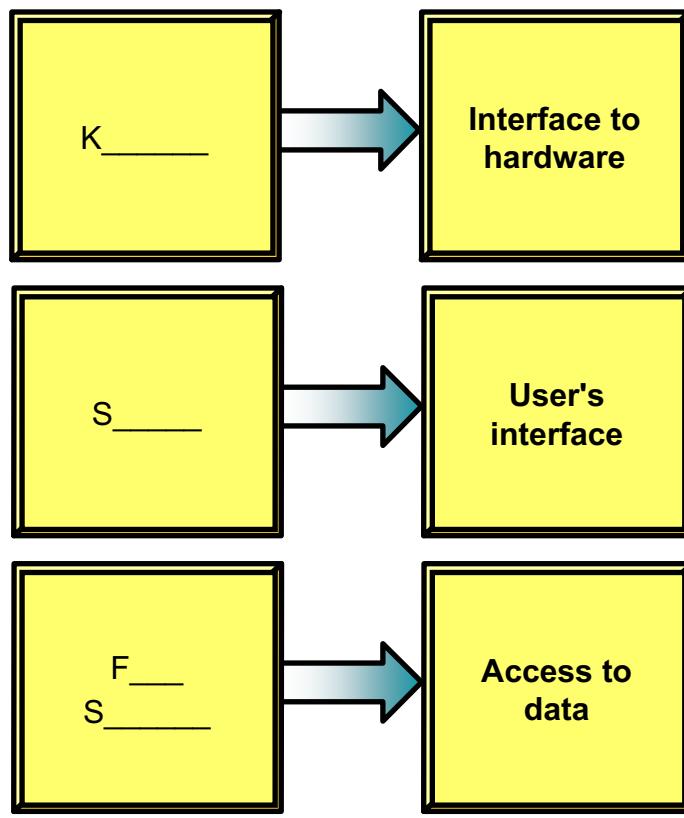


Figure 1-5. Activity: Fill in the Blanks

AU1310.0

Notes:

Operating system components

It is very important that you be able to identify the most important components of an operating system.

This visual introduces these components, but as you notice, the visual is not complete. Take some time and try to fill in the missing words.

Instructor Notes:

Purpose — To describe the meaning of the kernel, the shell, and file systems.

Details — Give the students some time to think about the missing words.

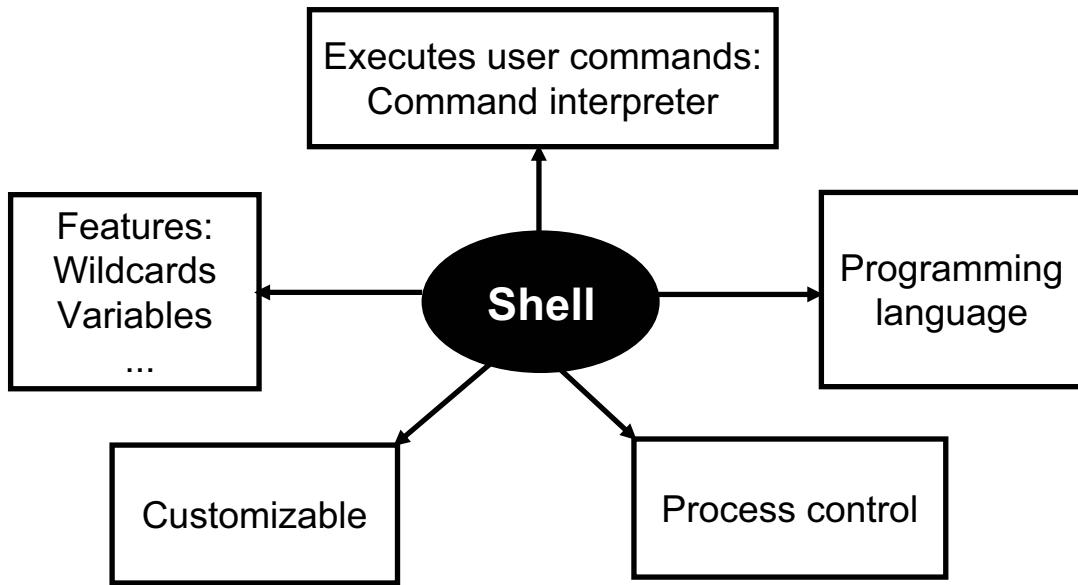
The missing words are:

1. Kernel
2. Shell
3. File system

Additional Information —

Transition Statement — Let's expand on a few of the important components, starting with the shell.

The Shell: User Interface to AIX



© Copyright IBM Corporation 2008

Figure 1-6. The Shell: User Interface to AIX

AU1310.0

Notes:

Introduction

When you log in successfully to an AIX system, a special program is started for you: *the shell*.

The shell

The shell waits for input and executes the commands and programs you type in. In other words the shell is a command interpreter.

The shell offers many features (like wildcards to match file names, variables, command line editing) that help the user in his daily work. We will discuss all these features in this course.

The shell offers different ways to control processes. In this course, we explain how a user can control his processes.

Customization

The shell is customizable. That means the user interface may be tailored according to the requirements of each user. Customizing the user environment is another topic in this course.

Besides all these properties, the shell is a programming language. You can write shell scripts to create and tailor commands. Writing simple shell scripts will be covered later in this course.

Instructor Notes:

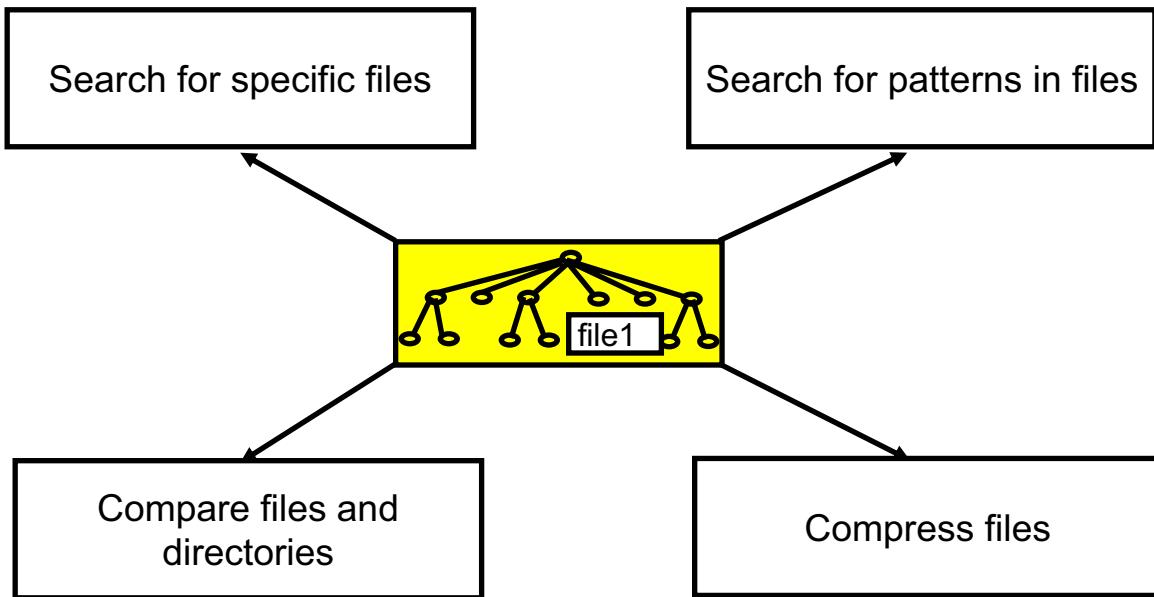
Purpose — Explain the term *shell*.

Details —

Additional Information —

Transition Statement — Let's now turn our attention to different AIX utilities.

Useful AIX Utilities



© Copyright IBM Corporation 2008

Figure 1-7. Useful AIX Utilities

AU1310.0

Notes:

AIX utilities

Two components that you use on AIX are *files* and *directories*. To work with these components, AIX offers a wide range of utilities:

- The **find** command to search for specific files
- The **grep** command to search for patterns in files
- Commands to compare files and directories
- Commands to compress and uncompress files to save disk space

Note that this list is not complete. Besides these utilities, the course introduces additional tools that are useful for your work.

Instructor Notes:

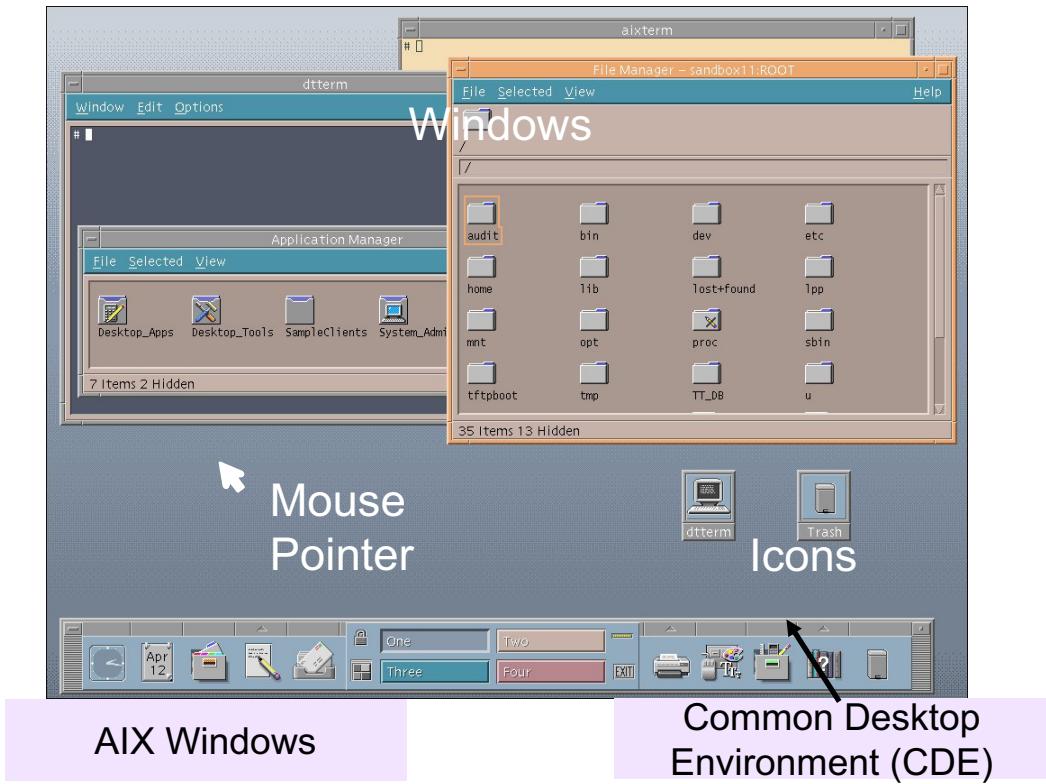
Purpose — Explain that this course introduces a wide range of utilities.

Details — We will cover the description of important directories of the filesystem later in this course.

Additional Information —

Transition Statement — Let's introduce the graphical user environments that come with AIX.

AIX Graphical User Interfaces



© Copyright IBM Corporation 2008

Figure 1-8. AIX Graphical User Interfaces

AU1310.0

Notes:

Graphical user interfaces

Modern operating systems are based on graphical desktops. These desktops consist of multiple *windows* where you can start different applications, *icons* that are minimized windows to manage the screen space, and further controls.

To execute certain actions on the desktop, you have to use the *mouse* attached to the system.

AIX offers two different graphical user interfaces:

- AIXwindows
- Common Desktop Environment (CDE)

Information on using and customizing these desktops is included in the appendix of this course.

Additional user interfaces

In AIX 5L, if you add the *AIX Toolbox for Linux Applications*, you can install two other graphical user interfaces:

- KDE
- GNOME

Instructor Notes:

Purpose — Introduce the desktops that are explained in the course.

Details —

Additional Information —

Transition Statement — We have no lab in this unit; therefore let's do the checkpoint.

Checkpoint

1. Which part of the operating system interacts directly with the hardware?
2. Which part of the operating system does the user interact with?
 - a) Shell
 - b) Kernel
3. Which editor is available across most UNIX platforms?
4. Write down the names of two AIX graphical user interfaces:
5. True or false: AIX only supports file systems on hard disks.

© Copyright IBM Corporation 2008

Figure 1-9. Checkpoint

AU1310.0

Notes:

Take some time and try to answer the questions.

Instructor Notes:

Purpose — To test the students' understanding of the presented material.

Details —

Checkpoint Solutions

1. Which part of the operating system interacts directly with the hardware? **Kernel**
2. Which part of the operating system does the user interact with?
 - a) **Shell**
 - b) Kernel
3. Which editor is available across most UNIX platforms? **vi**
4. Write down the names of two AIX graphical user interfaces:
 - a) **AIXwindows**
 - b) **Common Desktop Environment (CDE)**
5. True or false: AIX only supports file systems on hard disks.
False. AIX supports disk file systems, CD-ROM file systems, and network file systems.

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — Let's wrap up this unit with a quick summary.

Unit Summary

- The AIX Kernel interfaces to hardware devices and controls processes running in the AIX system.
- The user's interface to AIX is the shell. The shell is a command interpreter that offers great flexibility.
- To store data AIX uses a hierarchical file tree that consists of directories and files.
- AIX offers a wide range of useful utilities.

© Copyright IBM Corporation 2008

Figure 1-10. Unit Summary

AU1310.0

Notes:

Instructor Notes:

Purpose — To review the unit's key points.

Details — Use this visual to summarize the key points from the unit.

Additional Information —

Transition Statement — Let's move on to the next unit.

Unit 2. Using the System

Estimated Time

00:45

What This Unit Is About

This unit introduces the students to a few basic AIX commands.

What You Should Be Able to Do

After completing this unit, students should be able to:

- Log in and out of the system
- State the structure of AIX commands
- Execute basic AIX commands
- Use AIX commands to communicate with other users

How You Will Check Your Progress

Accountability:

- Checkpoint questions
- Exercise

Unit Objectives

After completing this unit, you should be able to:

- Log in and out of the system
- State the structure of AIX commands
- Execute basic AIX commands
- Use AIX commands to communicate with other users

© Copyright IBM Corporation 2008

Figure 2-1. Unit Objectives

AU1310.0

Notes:

Instructor Notes:

Purpose — Introduce the objectives for this unit.

Details —

Additional Information —

Transition Statement — Before any work can be done, you must gain access to the system.

Logging In and Out

- To Log in:

```
login: team01  
team01's Password: (the password does not appear)  
$
```

- To Log out:

```
$ <Ctrl-d>          (or)  
$ exit              (or)  
$ logout  
login:
```

© Copyright IBM Corporation 2008

Figure 2-2. Logging In and Out

AU1310.0

Notes:

Introduction

Because AIX is designed as a multi-user system, a level of security is implemented to control access. Each user of the system has a user name and associated password (optional).

User name

When the system has started and is ready for a user to log in, the login prompt (typically the word *login:*) is presented on the screen. At that point, the user should enter the supplied user name.

User password

If the user name requires a password, the system will prompt for the password in a similar manner. While the user is typing a password, it does not appear on the screen. It is highly recommended to use passwords on all user accounts.

If the user password was set up by the system administrator, the first time that the user logs in to the system, the user will be prompted to change their password.

Successful login

When logged in, the user is presented with a prompt (normally a dollar sign) which is the shell's way of requesting a command.

Exiting the system

To terminate the session the user may either enter the `exit` or `logout` command, or press the key combination `<Ctrl+d>` (holding down the `Ctrl` key while pressing the `d` key). The `<Ctrl+d>` combination generates the “end of file” (EOF) or “end of transmission” (EOT) character.

`logout` only works if you are in your login shell.

When the user logs out, after a few seconds a new login prompt will appear on the screen.

Instructor Notes:

Purpose — To describe the login and logout process.

Details — A user must supply at the minimum, a valid user name. If a password has been set up for the user, then they will be prompted for this as well.

If the user correctly enters all pieces of information during the login phase, then the user will be logged in to the shell/application (depending on setup).

Some users may instead log in through CDE in their own work environments. The use of CDE is covered in the appendix.

Additional Information — The login prompt that is displayed can be modified by the system administrator to reflect your own company name and copyright information (and anything else that is desired).

Transition Statement — Once you have logged in, one of the very first things that you might want to do is to change your password. Let's look at the command which enables this.

Passwords

- Creating or Changing:

```
$ passwd
Changing password for "team01"
team01's Old password:
team01's New password:
Enter the new password again:
$
```

© Copyright IBM Corporation 2008

Figure 2-3. Passwords

AU1310.0

Notes:

Changing the user password

The user password is the primary mechanism for ensuring security on an AIX system. All passwords are encrypted and cannot be decoded by other users.

The **passwd** command is used to change the user password and is an example of a simple command which may be entered at the shell prompt.

The system will start the **passwd** process which will prompt the user for their old password first. The characters typed are not displayed to the screen. To prevent users being locked out of the system through a simple typing error, the new password is entered twice. Only if the two entries match is the new password accepted. The old password is invalid thereafter.

When the **passwd** process terminates, the user is again presented with the prompt requesting another command.

Instructor Notes:

Purpose — To demonstrate how users can change their own passwords.

Details — As an ordinary user you can only change your own password, which is very desirable.

The `passwd` program is interactive and will prompt you twice for your new password.

All passwords are kept on the system in their encrypted form in the `/etc/security/passwd` file.

If normal users forget their passwords, then new passwords have to be generated by the system administrator. As the administrator has only the encrypted form of the passwords, they cannot tell you what your previous password was set to (this is an extra level of desired security). The only operation they can do is to reset the value.

It is up to the system administrator to enforce password restrictions, since by default, few restrictions exist. This information is discussed in the *AIX System Administration* class (AU14/Q1314).

Discussion Items — What password restrictions are enforced in your environment?

Examples of restrictions which might be discussed are: password lengths, minimum number of alphabetic or numeric characters, repeats of previous passwords, and prohibited passwords.

Additional Information —

Transition Statement — Let's look at the format of AIX commands.

Command Format

AIX commands have the following format:

\$ command *option(s)* *argument(s)*

```
$ ls  
$ ls -l  
$ ls /dev  
$ ls -l /dev
```

© Copyright IBM Corporation 2008

Figure 2-4. Command Format

AU1310.0

Notes:

Formatting a command

The order and separation of the elements of a command are important. The command or process name must come first. Spaces are used by the shell as separators on the command line and should not be placed within the command name.

Command options

The options should follow the command name, separated by a space, and preceded by a minus sign (-). Usually, multiple options may be grouped together immediately after a single minus sign or separated by spaces and each preceded by a minus sign. Options are typically used to modify the operation of the process.

Command arguments

The arguments follow the options, again separated by spaces. The order of the arguments will depend on the command.

Example

All three elements are not required to be present at all times, for example:

\$ ls	just command
\$ ls -l	command and option
\$ ls /tmp	command and argument

Instructor Notes:

Purpose — To define the syntax of AIX commands.

Details — Ensure first that the audience is aware of the differences between the command, the option (or flags as it is referred to in the documentation), and the argument. Then explain how these are represented in AIX; namely the first item is always treated as the command, anything following the command beginning with the - are treated as options, and any items not beginning with the - are treated as arguments. Also explain the significance of spaces, used as the delimiter, in AIX.

Explain the importance of the order of the syntax and how it varies from other operating systems, such as DOS, where the options are placed after the argument.

Additional Information — Reassure the students that the ls command with the -l flag will be discussed in this session; however, if you feel it necessary then give a brief description of how the command works.

Transition Statement — Let's look at some examples, to ensure that the concepts are clear for all.

Command Format Examples

WRONG:	RIGHT:
1. Separation: \$ mail - f newmail \$ who-u	1. Separation: \$ mail -f newmail \$ who -u
2. Order: \$ mail newmail -f \$ team01 mail \$ -u who	2. Order: \$ mail -f newmail \$ mail team01 \$ who -u
3. Multiple Options: \$ who -m-u \$ who -m u	3. Multiple Options: \$ who -m -u \$ who -mu
4. Multiple Arguments: \$ mail team01team02	4. Multiple Arguments: \$ mail team01 team02

THERE ARE EXCEPTIONS!!

© Copyright IBM Corporation 2008

Figure 2-5. Command Format Examples

AU1310.0

Notes:

Introduction

The commands in the visual display some examples of correct and incorrect command formats.

Exceptions

A few commands in AIX do not adhere to the common command format. Commands such as **tar**, **date**, and **ps** accept arguments that do not begin with a minus sign (-). This is to ensure backwards compatibility with older implementations of UNIX.

```
$ tar cvf /dev/rmt0 /home/team01/*
$ ps aux
$ date +%D
```

Instructor Notes:

Purpose — To ensure that the student is familiar with the command syntax rules by analyzing incorrect versions of commands.

Details — Before going through the visual, cover the right hand column of correct answers, and use this visual as a discussion topic, asking the students what is incorrect in each example.

Warning: The last example, that is, `mail team01team02` might be considered correct if there were a user defined on the system called `team01team02`.

It might be an aid to the student to mention briefly what the new commands and their options are, reassuring them that all the commands will be considered later in the session.

The graphic mentions that there are exceptions. As an example, students will learn about the `who am i` command in this course. From a system administrator's standpoint, the `date` and `ps` commands can use symbols other than the dash as options.

Additional Information —

Transition Statement — Now that we are familiar with the AIX command syntax, let's look at the `date` and `cal` commands.

The date and cal Commands

- Checking the date:

```
$ date  
Wed Nov 14 10:15:00 GMT 2007  
$
```

- Looking at a month:

```
$ cal 1 2003  
                 January 2003  
Sun   Mon   Tue   Wed   Thu   Fri   Sat  
       1      2      3      4  
  5      6      7      8      9     10    11  
12     13     14     15     16     17    18  
19     20     21     22     23     24    25  
26     27     28     29     30     31
```

- Looking at a year:

```
$ cal 2007
```

© Copyright IBM Corporation 2008

Figure 2-6. The **date** and **cal** Commands

AU1310.0

Notes:

Introduction

The visual shows how the **date** and **cal** commands can be executed.

Instructor Notes:

Purpose — To illustrate how the `date` and the `cal` commands can be executed.

Details —**date:**

Set globally by the system administrator and can be customized in a number of different formats. The command by default requires no options.

cal:

Syntax of the command: `cal [month] [year]` where month and year are optional fields. Also point out that if only one number is specified then it will be treated as the year. If no arguments are entered, the current month will be displayed.

Additional Information —

Transition Statement — Let's explain three other commands: `clear`, `echo`, and `banner`.

The clear, echo, and banner Commands

- **clear:** Clears the terminal screen

```
$ clear
```

- **echo:** Writes what follows to the screen

```
$ echo Lunch is at 12:00  
Lunch is at 12:00
```

- **banner:** Writes character strings in large letters to the screen

```
$ banner Hello
```

© Copyright IBM Corporation 2008

Figure 2-7. The **clear**, **echo**, and **banner** Commands

AU1310.0

Notes:

More commands

This visual shows how the **clear**, **echo**, and **banner** commands work.

Note: Instead of **echo** you can use the **print** command:

```
$ print Lunch is at 12:00
```

Lunch is at 12:00

Instructor Notes:

Purpose — To demonstrate how the `clear` and the `echo` commands work.

Details — Some details on each command.

clear:

The command first checks the `TERM` environment variable for the terminal type. Next the `/usr/share/lib/terminfo` directory, which contains the terminal definition files, is checked to determine how to clear the screen. If the `TERM` variable is not set or if it is set incorrectly, the command does not take any action.

echo:

Useful when used within shell scripts, for example, the prompt that is displayed uses the `echo` command to display the text as well as the new lines.

You can use the following escape conventions with the command (this is not an exhaustive list):

- `\b` displays a backspace character
- `\n` displays a newline character
- `\t` displays a tab character

banner:

Each line in the output can be up to 10 uppercase or lowercase characters in length. On output, all characters appear in uppercase with the lowercase input characters appearing smaller than the uppercase input characters.

To display more than one word on a line, enclose the text in quotation marks ("sample").

Additional Information — For further information look in the manual pages for the commands.

Transition Statement — Let's do a short activity so that students can process what they have learned.

Activity: Questions and Answers

1. What's wrong with the following commands?

\$ du -s k _____

\$ df-k _____

\$ du -a-k _____

2. Which command ...

... changes your password? _____

... clears the screen? _____

... prints out the current system date? _____

... exits the current shell? _____

© Copyright IBM Corporation 2008

Figure 2-8. Activity: Questions and Answers

AU1310.0

Notes:

Questions

Take some time and answer the questions.

Instructor Notes:

Purpose — Processing phase.

Details — Give the students some time to answer the questions. Afterwards review the questions.

1. What's wrong with the following commands?

`$ du -s k`

Correct Answer:

Multiple options must be grouped immediately after the minus sign. There should not be a space between the options. (Correct format: `du -sk`)

`$ df-k`

Correct Answer:

The option is not separated by a space from the command. (Correct format: `df -k`)

`$ du -a-k`

Correct Answer:

When multiple options are each preceded by a minus sign, they must be separated by a space. (Correct format: `du -a -k`)

2. Which command ... changes your password?

Correct Answer:

`passwd`

... clears the screen?

Correct Answer:

`clear`

... prints out the current system date?

Correct Answer:

`date`

... exits the current shell?

Correct Answer:

`exit`

Additional Information — The students do not need to know about the `du` and `df` commands at this point. These are just being used as examples. But if asked, explain that the `du` command reports on disk space usage and the `df` command displays information about total space and available space on a file system.

Transition Statement — Introduce the `who` and `finger` commands.

The who and finger Commands

- Finding who is on the system:

```
$ who
root          lft0          Sept 4 14:29
team01        pts/0         Sept 4 17:21
```

- Finding who you are:

```
$ who am i
team01        pts/0  Sept 4 17:21
$ whoami
team01
```

- Displaying information about the users currently logged on:

```
$ finger team02
Login name: team02
Directory: /home/team02                         Shell:
/usr/bin/ksh
On since Mar 04 16:17:10 on tty3
No Plan.
```

© Copyright IBM Corporation 2008

Figure 2-9. The **who** and **finger** Commands

AU1310.0

Notes:

who command

The **who** command identified who is logged in and where they have logged in from. Sometimes it is desirable to know what terminal you are working with, which can easily be identified with the **who am i** command. This will produce output similar to the **who** command but only from your own login session.

Earlier in the unit, options were introduced with the **who** command. Here are some more details on their functions:

- u displays the user name, extended workstation name, login time, line activity and process ID of the current user.
- m displays information about the current terminal and this is equivalent to the **who am i** command.

finger command

The **finger** command has a default format which displays: Full user name, login time, user's \$HOME directory and user's login shell.

You can use your own username with the **finger** command to find out information about yourself.

Instructor Notes:

Purpose — To identify who else is logged in to the system and how to obtain further information about a user.

Details — Describe the format of the output, and mention what each of the fields represents.

Additional Information — As the output of the **who** command shows the **root** user logged in to the terminal **lft0**, you could give a brief description of the importance and the power of the **root** user ID. Also note that the user **team01** is logged in to a terminal with an ID of **pts/0**. This is an example of a user running AIX from a PC emulation program (such as Hummingbird Exceed). Students will be accessing AIX in this way from a number of our classrooms. It should be made clear that pseudo terminals are mainly for graphic windows running terminal emulations, and may have nothing to do with a remote workstation connecting in to this machine.

The **who -m** or **who am i** command reads original login information from the **/etc/utmp** file. The **whoami** command does not. If you switch your user ID via the **su** utility, the commands will report different results.

```
$ whoami  
team01  
$ who am i  
team01 pts/0 Nov 13 14:05 (192.168.1.100)  
$ su - newuser  
newuser's Password:  
$ whoami  
newuser  
$ who am i  
team01 pts/0 Nov 13 14:05 (192.168.1.100)
```

You might want to describe also (depending on the audience) how the **\$HOME/.plan** and **\$HOME/.project** files can be used to display additional information with the **finger** command.

Remember we have not discussed what **\$HOME** is so a brief description might be needed. If these files exist, their contents will also be displayed as part of the output of the **finger** command.

This command can also be used to look up information about users on a remote system, but you must know the machine name on which they are defined.

Information added to the full name (information) field in the **/etc/passwd** file will also be displayed with the **finger** command. If commas are used to separate the information, all information before the first comma is shown by the **finger** command under the heading **in real life**. Everything after the first comma is shown under the heading **site info**.

Transition Statement — Now that we know who is logged in, let's see how we can send messages to one another.

Sending Mail

```
$ mail team01
Subject: Meeting
There will be a brief announcement meeting today
in room 602 at noon.
<Ctrl-d>
Cc: <Enter>
$
```

```
$ mail team20@sys2
Subject: Don't Forget!
Don't forget about the meeting today!
<Ctrl-d>
Cc: <Enter>
$
```

© Copyright IBM Corporation 2008

Figure 2-10. Sending Mail

AU1310.0

Notes:

Introduction

The mail command is an interactive command used to send and receive mail messages.

Sending a message

To send a message, invoke the command by passing it valid user IDs. If more than one name is given, the names must be separated from each other with a blank space.

Next the prompt `Subject:` will automatically be displayed. The sender should fill in this field with one line of text which closely describes the contents of the mail body. This is the line which will appear in the recipient's list of incoming mail.

After the subject line, the note body should then be entered, and once complete, press a `<Ctrl-d>` on the next available blank line.

Note: This must be the first and only character on that line. This is the end of file marker.

The **Cc:** prompt (denoting carbon copy) will then be displayed, which can be left blank, or a string of user IDs can be entered.

After the last prompt, the shell prompt should be displayed.

Sending mail to other systems

When sending mail to another user on your same system, enter `mail <username>`. To send mail to a user on another computer system, it is necessary to indicate the name (the host name) of that computer. For example,

`mail <username>@<hostname>`

Instructor Notes:

Purpose — To determine how messages can be sent from one user name to another.

Details —

Additional Information —

Transition Statement — We have just considered the `mail` command for sending a note. The same command can be used to receive the incoming mail. Let's see how this is achieved.

Receiving Mail

[YOU HAVE NEW MAIL]

\$ mail

```
Mail [5.2 UCB] [AIX5.X] Type ? for help
"/var/spool/mail/team01": 2 messages 1 new
U 1 team05      Tue Jan  7 10:50   10/267 "Hello !"
>N 2 team02      Wed Jan  8 11:25   16/311 "Meeting"
```

? t 2

From team02 Wed Jan 8 11:25 2003

Date: Wed 8 Jan 2003 11:25

From: team02

To: team01

Subject: Meeting

Cc:

There will be a brief announcement meeting today in room 602 at noon.

? d (Delete message)

? q (Quit mail command)

© Copyright IBM Corporation 2008

Figure 2-11. Receiving Mail

AU1310.0

Notes:

Introduction

The user is informed that new mail items have arrived when the [YOU HAVE NEW MAIL] message is displayed. This message does not get automatically displayed as soon as the incoming mail arrives. The shell does a check on all the mailboxes by default once every 600 seconds. If it detects a new piece of mail, then it displays the message (which itself can be customized by the system administrator).

Receiving mail

To receive the mail items use the **mail** command without any options. It will list header information and a one line description for each unread item followed by the prompt ?. This is different from the shell prompt. AIX uses the ? as the mail subsystem prompt.

Controlling the mail subsystem

At this prompt, the user may enter any of the mail subsystem commands. To obtain a list enter a **?** at the prompt. Normal operations like saving, deleting, viewing, and so forth, can be carried out on each mail item.

Some of the commands that can be used at the **?** prompt are:

- d** delete messages
- m** forward messages
- R** send reply to sender of messages in the queue
- q** exit mail and leave messages in the queue
- s** append messages to a file
- t** display a message
- x** exit mail and discard all changes to the queue

Certain subcommands like “**d**” and “**t**” accept message numbers as arguments. They default to the current message if none is specified.

There are many more. To obtain a the list of commands available type in a **?** at the **?** prompt or see the *AIX 6.1 Commands Reference Manual*.

Leaving the mail subsystem

Having finished working with the mail items, to return to the shell prompt, you must enter a **q** (for quit) at the **?** prompt. This will take you out of the mail subsystem.

If you wish to quit the mail subsystem and discard any changes to the queue, enter an **x** (for exit) at the prompt. Messages that were marked for deletion are not deleted.

Any saved mail items which have been read but not deleted cannot be viewed again using the above method. Once the mail item is read, it will be stored in a file in the user's home directory called **\$HOME/mbox**.

To view these items you must use the **mail -f** command. This will look at your default mailbox. If you have created other mailboxes, then you have to also specify the mailbox name.

Instructor Notes:

Purpose — To define how incoming mail can be accessed and manipulated.

Details — Explain that the example in the visual shows two messages in the mailbox, one of which is new. Also explain each character or symbol that is part of the one line description for each mail item, for example:

- > displays the current messages
- U unread mail which must have been in the mailbox previously
- N new mail which has just been received

The default time that the shell checks for new mail can be customized by the system administrator by modifying the MAILCHECK variable.

You may want to explain that the `mail` command behaves differently depending on whether you pass arguments to it or not. Invoking the command without arguments invokes full interactive mode, while supplying arguments (like subject, cc, and recipient) allows it to execute a specific task (like sending mail).

Additional Information — For further information look in the manual pages of the AIX 6.1 online documentation.

Transition Statement — Let's look at some other useful commands for communicating with users.

The write and wall Commands

- Send **messages** to other **users** on a system
 - write** provides *conversation-like* communication with another logged-in user. Each user alternatively sends and receives messages.

```
$ write team01          (or)
$ write sarah@moon
```

- The **wall** command writes to all terminals. This is useful to notify all users of a system event:

```
$ wall The system will be inactive from 10pm today.

Broadcast message from team01@sys1 (pts/1) at 15:11:58

The system will be inactive from 10pm today.

$
```

© Copyright IBM Corporation 2008

Figure 2-12. The **write** and **wall** Commands

AU1310.0

Notes:

write and wall commands

The **write** command can be used to send messages to users on this system as well as users of other systems connected to the network.

Both **write** and **wall** will only send messages to users that are logged in. By default, all users have the ability to execute the **wall** command.

write sends messages to a single user. **wall** sends messages all users currently logged in to the system.

Receiving messages

For a user to receive a message, that user must be logged in and must not have refused permission.

Write example

To hold a conversation using **write** enter:

```
$ write sam
```

Press **<enter>** and type:

I will need to re-boot the system at noon.**<enter>**

○ **<enter>**

This starts the conversation. The ○ at the beginning of the next line means the message is over and you are waiting for a response. Now, Sam enters:

```
$ write bill
```

Thank you for letting me know! **<enter>**

○○ **<enter>**

The ○○ means “over and out” telling the sender you have nothing more to say. Press **<Ctrl+d>** to end the write session.

Sending messages to users on other systems

write can also be used across a network as long as the **writesrv** daemon is running. To use **write** over the network type **write <username>@<hostname>**.

Instructor Notes:

Purpose — Explain how to send messages to system users.

Details — To use `write` to send a prepared file, use redirection: `write bill < message.text`. Students have not yet learned about redirection, so do not bring this up unless asked.

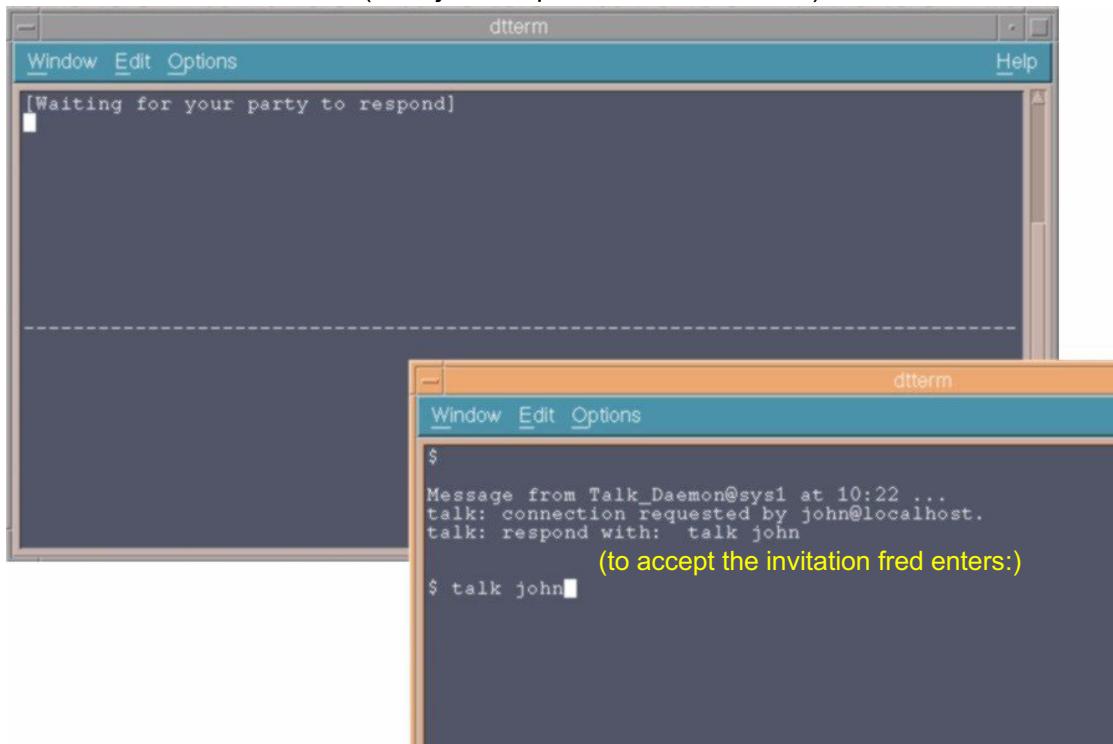
If a user is logged in more than once, `write` will display the message on the first login terminal.

Additional Information — The “o” and the “oo” strings are not subcommands to the write facility. They are just a common convention for communicating your intent to your write partner. You could just as well type out a phrase such as “your turn” or “done now, gotta go.”

Transition Statement — We can also establish a two-way communication with users.

talk with Another User

\$ talk fred (user john requests the connection)



© Copyright IBM Corporation 2008

Figure 2-13. talk with Another User

AU1310.0

Notes:

Two-way communication

The **talk** command allows two users to hold a conversation. One user invites the other to hold a conversation by issuing the **talk** command. The **talk** command opens a send window and a receive window on each user's display. Each user is then able to type into the send window while the **talk** command displays what the other user is typing.

If the invitation is accepted, each user's screen is split horizontally into two windows. In the top window everything the other user types is displayed.

To close the connection, press the INTERRUPT key **<Ctrl+c>**.

talk can also be used in a network. To talk to **fred** on **sys1**, the command would be **talk fred@sys1**.

Instructor Notes:

Purpose — Demonstrate how the `talk` utility works.

Details — If a user is logged in more than once, the `talk` utility will use the first terminal session, ignoring any others.

The slide and notes show the `talk` command opening a graphical window on the other end. Point out that this also works with ASCII terminals.

Additional Information —

Transition Statement — There may be times when you wish to deny other users the ability to write messages to your screen. Let us see how this is done.

mesg

- The **mesg** command controls whether other users on the system can send messages to you:

```
team01$ mesg  
The current status is y.  
team01$ mesg n  
team01$
```

```
team02$ write team01  
write: 0803-031 Permission denied.  
team02$
```

© Copyright IBM Corporation 2008

Figure 2-14. **mesg**

AU1310.0

Notes:

Permitting messages

The shell startup process permits messages by default. The visual shows how the **mesg** command work can be used to allow or deny messages.

The **mesg** command determines whether messages can be sent to the user with either the **talk**, the **write**, or the **wall** commands.

Permitting or denying messages can also be set as part of your session customization which we will cover later in this course.

Instructor Notes:

Purpose — Define how the `mesg` command works.

Details — Tell the students you will be teaching them how to customize their login session with the `.profile` file in a later unit. System administrators can send messages to all users regardless of these settings.

Additional Information — We should have a look back to this page within the activity in Unit 6. File Permissions.

Transition Statement — Let's show some keyboard tips.

Keyboard Tips

<Backspace>	Corrects mistakes
<Ctrl-c>	Terminates the current command and returns to the shell
<Ctrl-d>	End of transmission or end of file
<Ctrl-s>	Temporarily stops output to the screen
<Ctrl-q>	Resumes output (stopped by Ctrl-s)
<Ctrl-u>	Erases the entire line

© Copyright IBM Corporation 2008

Figure 2-15. Keyboard Tips

AU1310.0

Notes:

Keyboard tips

Do not use the cursor keys to correct mistakes, such as the up or down arrow key or the tab keys. The best way to correct mistakes is to use the **Backspace** key.

The <Ctrl-s> and <Ctrl-q> keys are somewhat system-dependent. On some ASCII terminals, the **Hold** key can be used as a toggle key to start and stop output to your terminal.

Instructor Notes:

Purpose — To aid the student when using the system (especially novice users doing the first exercise).

Details — Go through each key sequence, mentioning the importance of each and when they would use them. The list is intended to be an aid for the student just before the exercise is begun.

In the event a student's Backspace key does not work, `<Ctrl-h>` can also be used to backspace.

Discussion Items — When explaining the `<Ctrl-d>` key sequence, ask the students if they remember seeing this particular key sequence with any other command. They should remember the `mail` command to indicate the end of file.

One could mention accidentally pressing `<Ctrl-s>` instead of the intended uppercase S `<Shift-s>` is a common cause of having a hung terminal. It never hurts to try `<Ctrl-q>` when hung just in case this is the case

Transition Statement — Before we go to the lab, let's do a few checkpoint questions.

Checkpoint

1. What is the correct command syntax in AIX?
\$ mail newmail -f
\$ mail f newmail
\$ -f mail
\$ mail -f newmail
2. What command would you use to send mail items?
3. What are other commands that can be used to communicate with other users?
4. What output would you expect from the following command: **cal 8**?
5. Which command would you use to find out when a particular user logged in?
\$ who am i
\$ who
\$ finger everyone
\$ finger username

© Copyright IBM Corporation 2008

Figure 2-16. Checkpoint

AU1310.0

Notes:

Instructor Notes:

Purpose — Test the students' knowledge of the presented materials.

Details —

Checkpoint Solutions

- What is the correct command syntax in AIX?

```
$ mail newmail -f  
$ mail f newmail  
$ -f mail  
$ mail -f newmail
```

- What command would you use to send mail items? **mail username**
- What are other commands that can be used to communicate with other users? **talk, write, and wall**
- What output would you expect from the following command: **cal 8**?
The calendar for the year 8 AD
- Which command would you use to find out when a particular user logged in?
\$ who am i

```
$ who  
$ finger everyone  
$ finger username
```

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — Next step is a lab exercise.

Exercise: Using The System



© Copyright IBM Corporation 2008

Figure 2-17. Exercise: Using The System

AU1310.0

Notes:

Exercise Introduction

At the end of the lab, you should be able to:

- Log in to an AIX system and change passwords
- Execute basic commands
- Use the **wall** and **write** commands to communicate with other users
- Use keyboard control keys to control command line output

Instructor Notes:

Purpose — Transition to the exercise.

Details — Explain what students will learn in the exercise.

Discussion Items —

Transition Statement — Let's wrap up this unit before moving on to the next one.

Unit Summary

- AIX commands can use multiple options and arguments and must follow proper syntax rules.
- There are many simple, yet powerful commands such as:
`date`
`cal`
`who, who am I`
`finger`
`echo`
`clear`
`banner`
- Communicate with other UNIX users using commands such as: `mail`, `write`, `talk`, and `wall`.

© Copyright IBM Corporation 2008

Figure 2-18. Unit Summary

AU1310.0

Notes:

Instructor Notes:

Purpose — Summarize the key points from this unit.

Details —

Additional Information —

Transition Statement — Time for a new unit.

Unit 3. AIX 6.1 Documentation

Estimated Time

00:45

What This Unit Is About

This unit illustrates the different methods that can be used to obtain online help.

What You Should Be Able to Do

After completing this unit, students should be able to:

- Use the `man` command to view information about AIX commands
- Describe the use of AIX 6.1 Web-based documentation

How You Will Check Your Progress

Accountability:

- Checkpoint questions
- Exercise

Unit Objectives

After completing this unit, you should be able to:

- Use the **man** command to view information about AIX commands
- Describe the use of AIX 6.1 Web-based online documentation

© Copyright IBM Corporation 2008

Figure 3-1. Unit Objectives

AU1310.0

Notes:

Instructor Notes:

Purpose — To introduce the unit objectives.

Details —

Additional Information —

Transition Statement — When looking for help at the command line, you can use the `man` command to gather information about AIX commands.

man Command

- The **man** command provides reference information on **commands**, **subroutines**, and **files**
- Manual information consists of:

<ul style="list-style-type: none"> - Purpose - Syntax - Description - Flags - Examples - Files - Related Information 	<ul style="list-style-type: none"> (one line description) (all valid options and arguments) (verbose description) (description of all valid options) (command examples) (associated files and directories) (additional resources and information)
--	--

© Copyright IBM Corporation 2008

Figure 3-2. **man** Command

AU1310.0

Notes:

man command features

The **man** command will look in the online manual for information on the commands, subroutines, and files with the name title. This information will be presented on the screen one page at a time for the user to browse.

The information consists of:

Purpose	The title and a one line description of the command
Syntax	A list of all valid options and arguments
Description	Many pages of information about the function and usage of the command with examples
Flags	Description of available options
Examples	Samples of how to use the command
Files	Any system files associated with the command
Related Info.	The names of any related commands

Instructor Notes:

Purpose — To describe the main features of the `man` command.

Details — Explain why `man` would be used and how it is most useful to gain further information about a known command. It can provide information about commands that are specified by name or it can provide information on all commands whose descriptions contain a set of user-specified keywords (with the `-k` option that will be covered on a later visual).

The output from `man` is broken up into a number of sections, which have all been listed. Go through each section explaining what might be seen in each section.

The output from the `man` command is presented one page at a time. Use the space bar to move forward one page at a time, and the `Enter` key to move forward one line at a time. Also displayed is the percentage of the output already viewed. This behavior is due to the nature of the `more` command, which is the default viewer.

While you are viewing the output, you can use the `h` key to obtain a list of all the commands and key sequences that are available while viewing the output using `man`. `man` uses the variable `PAGER` to determine what viewer is used to display the man pages. If the `PAGER` variable is not set, the `more` command is used to view the `man` pages.

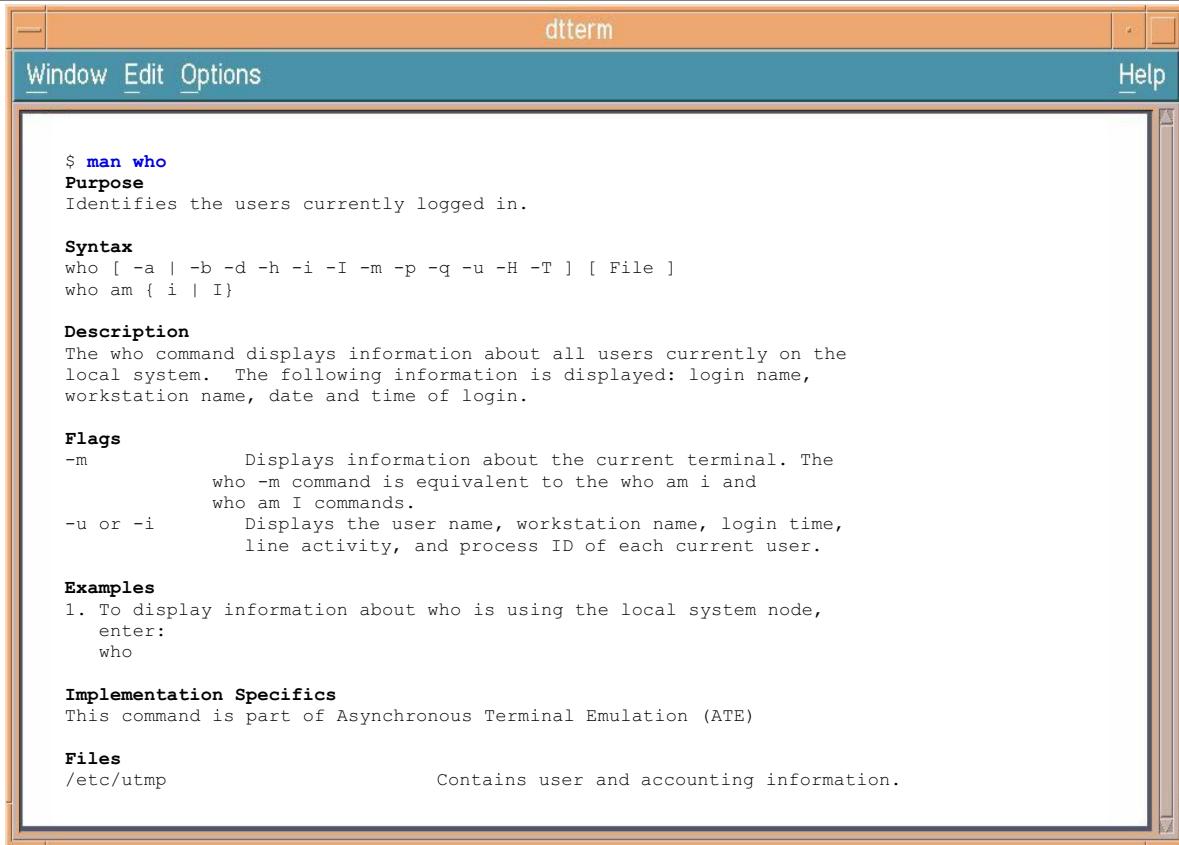
To break out of the `man` command use `<ctrl-c>`.

Additional Information — To obtain more information type in `man man`.

For background only: Prior to AIX V4.3, the `man` command extracted its information from the InfoExplorer database. Starting in AIX V4.3, this information is instead extracted from the AIX documentation which is now stored in HTML format. In terms of overall functionality, the `man` command works the same in all releases of AIX.

Transition Statement — Let's look at an example of the `man` command.

man Example



The screenshot shows a terminal window titled "dtterm". The menu bar includes "Window", "Edit", "Options", and "Help". The main pane displays the man page for the "who" command:

```
$ man who
Purpose
Identifies the users currently logged in.

Syntax
who [ -a | -b -d -h -i -I -m -p -q -u -H -T ] [ File ]
who am { i | I }

Description
The who command displays information about all users currently on the
local system. The following information is displayed: login name,
workstation name, date and time of login.

Flags
-m           Displays information about the current terminal. The
            who -m command is equivalent to the who am i and
            who am I commands.
-u or -i      Displays the user name, workstation name, login time,
            line activity, and process ID of each current user.

Examples
1. To display information about who is using the local system node,
   enter:
   who

Implementation Specifics
This command is part of Asynchronous Terminal Emulation (ATE)

Files
/etc/utmp      Contains user and accounting information.
```

Figure 3-3. man Example

AU1310.0

Notes:

This example shows the **man who** command. Note that this example has been condensed to fit on one page.

Instructor Notes:

Purpose — To illustrate an example using the `man` command.

Details — The example shows the `man who` command. Note that the students should be familiar with the `who` command from the previous unit.

Point out that each section has been condensed so that an example from each section is seen.

Go through the syntax diagram and ensure that students are familiar with all the symbols, such as:

[] optional field

a | b either select a or b

{ } mandatory field

Mention the usefulness of the examples section, as most times an example that closely matches a desired task can be found.

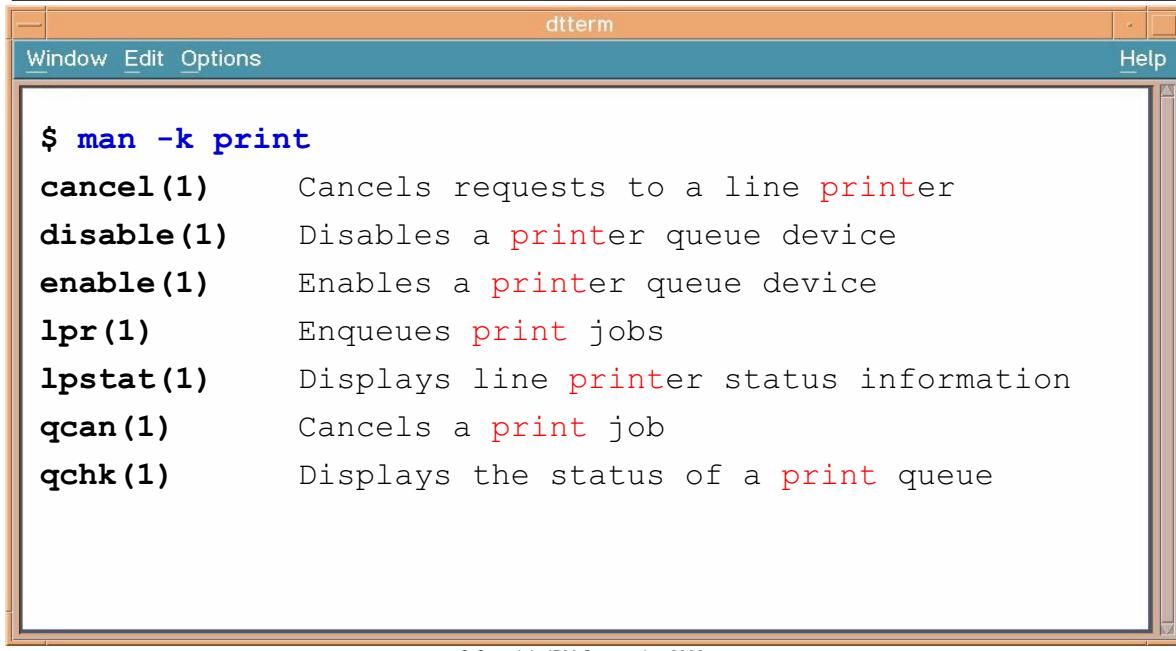
Additional Information —

Transition Statement — We have just seen how the `man` command can be used to search for a particular command. What if we are not sure about the command name and only know the topic area that the command belongs to; for instance, we cannot remember how to cancel a job from a print queue, but we know that the command involves printers. `man` provides the ability to search on a keyword.

man -k: Working with a Keyword

The **-k** option of the **man** command allows you to print out one line descriptions of all entries which match the given keyword

Example:



```
$ man -k print
cancel(1)      Cancels requests to a line printer
disable(1)     Disables a printer queue device
enable(1)      Enables a printer queue device
lpr(1)          Enqueues print jobs
lpstat(1)      Displays line printer status information
qcan(1)         Cancels a print job
qchk(1)         Displays the status of a print queue
```

© Copyright IBM Corporation 2008

Figure 3-4. **man -k**: Working with a Keyword

AU1310.0

Notes:

Enabling the -k feature

To use the **-k** flag, a superuser must have typed **catman -w** to create the **/usr/share/man/whatis** file.

man -k command

The **man -k** command shows the manual sections that contain any of the given keywords in their purpose section. The output from the command begins with the name of a command and the section number in which the command appears.

If you want to view the output from the command **enable(1)**, then you can enter **\$ man enable** to obtain the manual pages for the **enable** command. If the section number is omitted, the **man** command searches all the sections of the manual.

To obtain further information about the various **man** sections enter **man man**. Note that the **apropos** command is equivalent to **man -k**.

Instructor Notes:

Purpose — To demonstrate how `man` can be used to search on a keyword.

Details — Explain the output in the example and that the manual pages are broken up into a number of different sections (as indicated by the number after the command). To obtain a list of all the sections look in the manual pages of the `man` command (that is, `man man`).

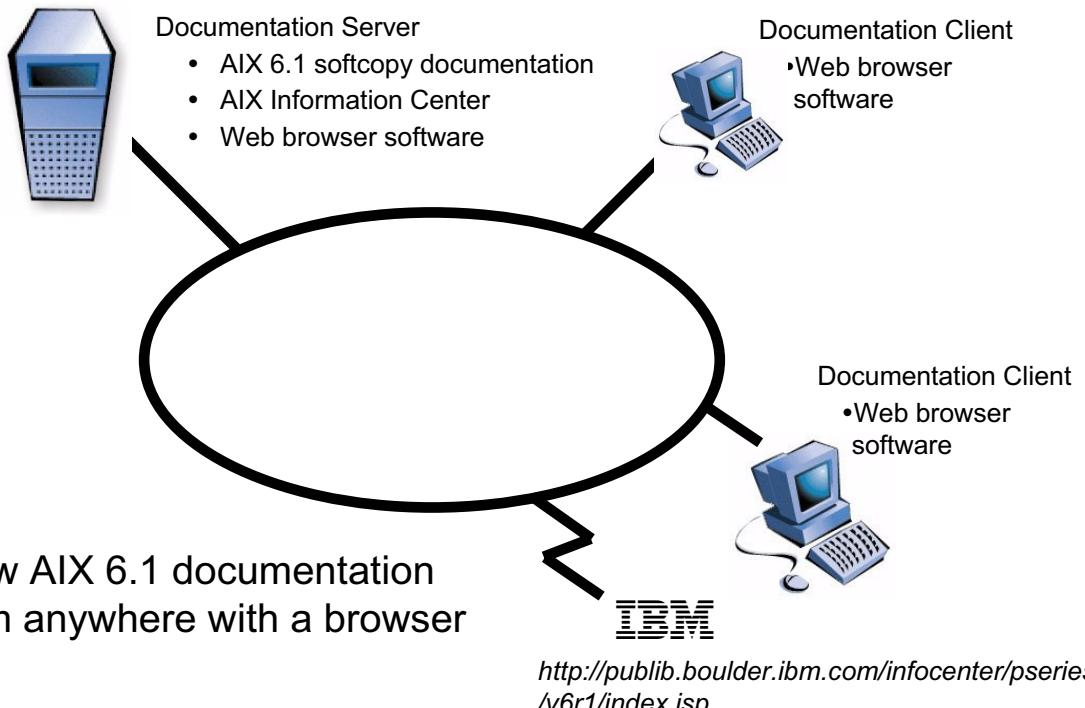
When you invoke the `man` command without the section number, if the command has references in more than one section, the output from all sections will be given.

Another command, `apropos`, can also be used and is equivalent to using the `man -k` command.

Additional Information —

Transition Statement — Now let's take a quick look at how online documentation is accessed in AIX 6.1.

Viewing AIX 6.1 Documentation



© Copyright IBM Corporation 2008

Figure 3-5. Viewing AIX 6.1 Documentation

AU1310.0

Notes:

Documentation server

In addition to providing `man` commands to make finding information easy, AIX also provides system manuals. The documents are available on the Internet at the IBM Web site <http://publib.boulder.ibm.com/infocenter/pseries/v6r1/index.jsp>. For sites without access to the Internet, softcopy documentation can be loaded on a *documentation server* within a private network. Any other computer in the network with Web browser software can then become a *documentation client* and access these documents from the server.

Requests for documents

When users on a client computer request an AIX document, the request is sent to the Web server on a documentation server which then sends back the requested item. When searches are performed, they are done on the server computer and the results are then sent back to the user on the client computer.

Instructor Notes:

Purpose — To provide the big picture of how AIX 6.1 documentation works.

Details — When AIX is installed, the system administrator might set up AIX softcopy documentation. IBM strongly suggests using documentation provided on the Internet, however some of your students may not have, or be allowed, external access to the Web.

Note, these students will not be installing the documentation, this information is being provided to help them understand that it can be found on their system or on the Internet.

The system administrator will first set up the *documentation server*. It has the following installed:

- The AIX 6.1 Documentation - This consists of most of the manuals that are found in the traditional InfoExplorer product.
- Web browser software - This is necessary if users on the server wish to access documents. Mozilla Firefox for AIX is available as a Web download from www.ibm.com/servers/aix/browsers and on the Mozilla Firefox for AIX CD that can be ordered with AIX. The Mozilla Web browser is also available as a Web download, and supports all levels of AIX 5L. Actually any browser can be used, assuming it is HTML 3.2-enabled and supports frames.
- In AIX 5L V5.2, the Documentation Library Service is installed by default with the base. Prior to AIX 5L V5.2, additional software must also be installed on the documentation server to support the search function. Starting with AIX 5L V5.3, the documentation is accessed from the AIX Information Center. The DLS is still supported with AIX 5L V5.3, but no new documents are being added. Support may be dropped in future releases.

The client system must have a Web browser installed.

Users at the client system will issue requests to view AIX documentation. These requests will be sent to the documentation server and the results will then be sent back to the user at the client.

If you have a stand-alone computer, both the server and client software are installed on the same stand-alone computer. Instead of going to a remote computer, requests from users on the stand-alone computer go to the Web server software on their own computer.

Be sure not to go into too much detail on the software that needs to be installed. In this class, the purpose here is for students to understand the big picture.

Additional Information —

Transition Statement — In order to access the AIX documentation online, a Web browser must be installed.

Accessing the Documents from a Web Browser

- Mozilla Firefox for AIX
 - New features, supports many current Web standards
 - 64-bit
 - Available as a Web download or on CD when ordering AIX (Mozilla Firefox for AIX CD)

- Mozilla
 - Standard browser beginning with AIX 5L 5.3
 - Supports all versions of AIX 5L
 - 32-bit
 - Available as a Web download

<http://www.ibm.com/servers/aix/browsers>

© Copyright IBM Corporation 2008

Figure 3-6. Accessing the Documents from a Web Browser

AU1310.0

Notes:

Web Browser

Mozilla Firefox for AIX is an open source Web browser. It delivers helpful new features and continues to lead the way in online security. It implements technologies like the Gecko layout engine, and supports Web standards or draft standards like HTML, XHTML, XML, CSS, DOM, and many more. Mozilla Firefox for AIX implements OJI, the Open Java Interface to AIX Java through the AIX Java Plug-in.

Mozilla Firefox version (64 bit) is available as a Web download and on the Mozilla Firefox for AIX CD that can be ordered with AIX 6.1.

The Mozilla Web browser was introduced with AIX 5L V5.2, and became the standard Web browser in AIX 5L V5.3. It is available as Web download.

Instructor Notes:

Purpose — Talk about how to access the documentation with Mozilla.

Details —

Additional Information —

Transition Statement — Once a Web browser is installed, it is time to install the documentation.

AIX 6.1 Documentation

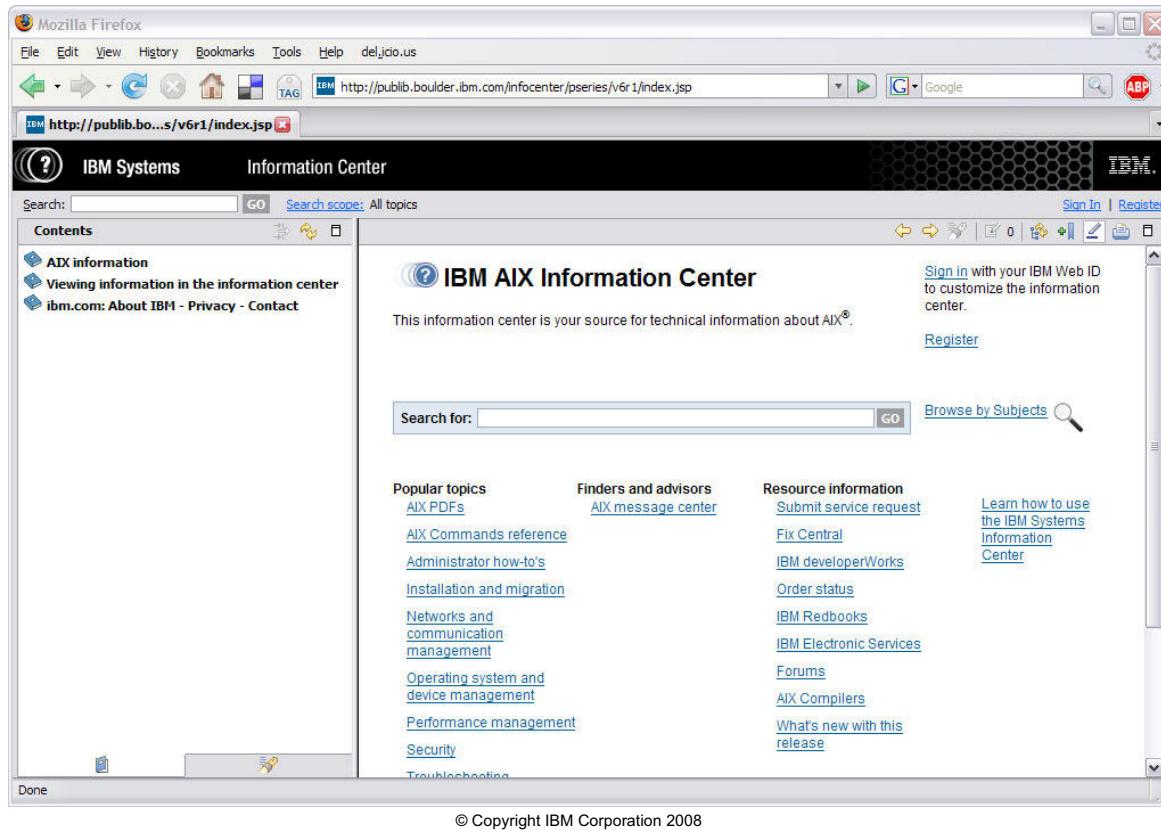


Figure 3-7. AIX 6.1 Documentation

AU1310.0

Notes:

Accessing documentation

Once the documentation is set up, it can be accessed from the AIXWindows or CDE environment with the **infocenter** command.

Web access

If the documentation was not installed on your system, it can also be accessed at:

<http://publib.boulder.ibm.com/infocenter/pseries/v6r1/index.jsp>

Viewing documents

The documents can be viewed two ways. Either by selecting the entire document with the PDF tag at the end of each document name or by selecting the HTML tag and viewing the document section by section.

Searching

In the top left corner of the Information Center page, there is a box for entering search strings. Entering information and selecting GO will search all documents for the string. You can select Search Scope to limit the search to a set of documents.

Printing documents

The Information Center allows you to print documents in two ways.

You can download the PDF document and print the entire document from Adobe Acrobat.

Or access a section of a document in HTML and print that section as you would normally print the contents of a Web page. Find the section you wish to print and use the browser's Print function usually found in the File menu.

Instructor Notes:

Purpose — Show the students how to access the online documentation once it has been configured.

Details — The Information Center is also available using CDE. It can be accessed using the CDE Help Manager or the Application Manager.

Additional Information — The documentation is designed to work in a graphical environment. You must have a browser and graphic monitor to see it.

ASCII users will have to rely on [man](#) pages.

Transition Statement — The online documentation has a search function.

Search AIX 6.1 Documentation

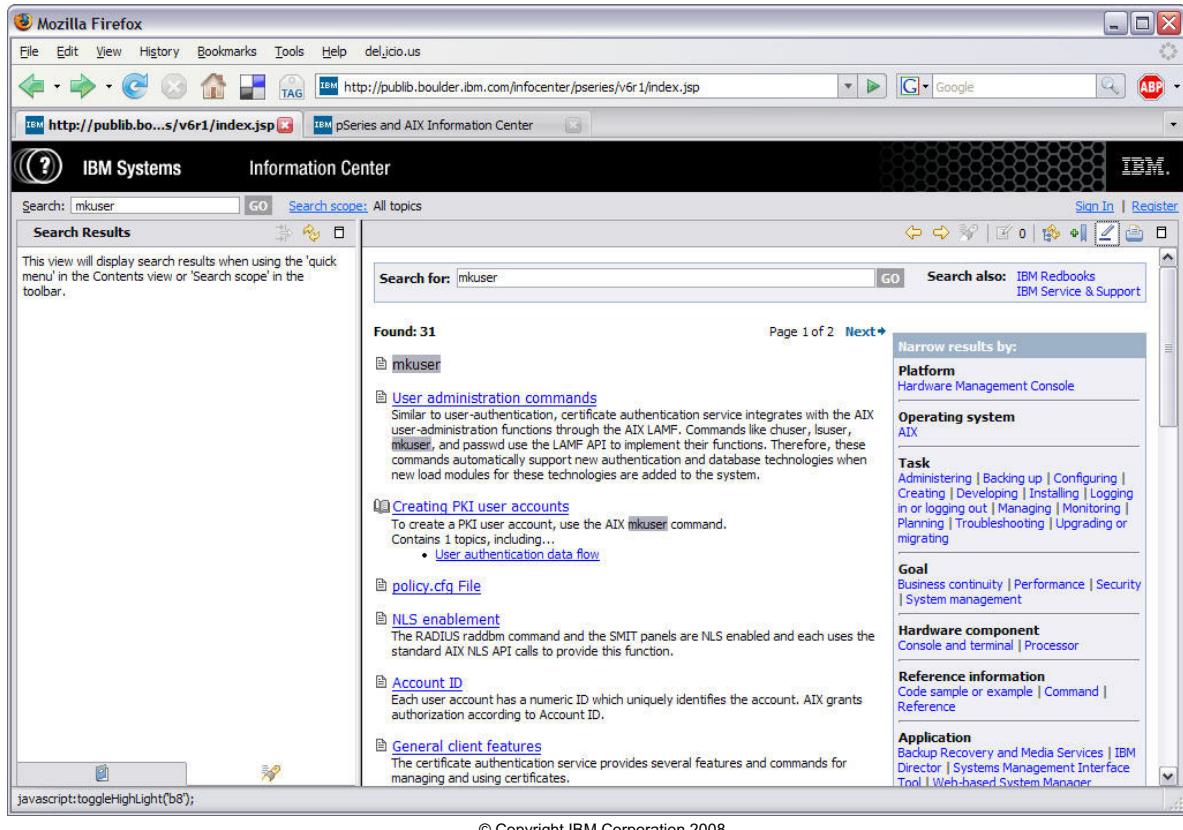


Figure 3-8. Search AIX 6.1 Documentation

AU1310.0

Notes:

Searching documents

Probably the easiest way to find an answer is to search the documentation using the Search window on the Information Center screen.

Above are the results of a search. In the right frame are the matches from the search in order of quality with the best match at the top of the list.

Instructor Notes:

Purpose — Describe how to search the AIX 6.1 documentation.

Details —

Additional Information —

Transition Statement — Let's take a look at the search scope feature.

Search Scope

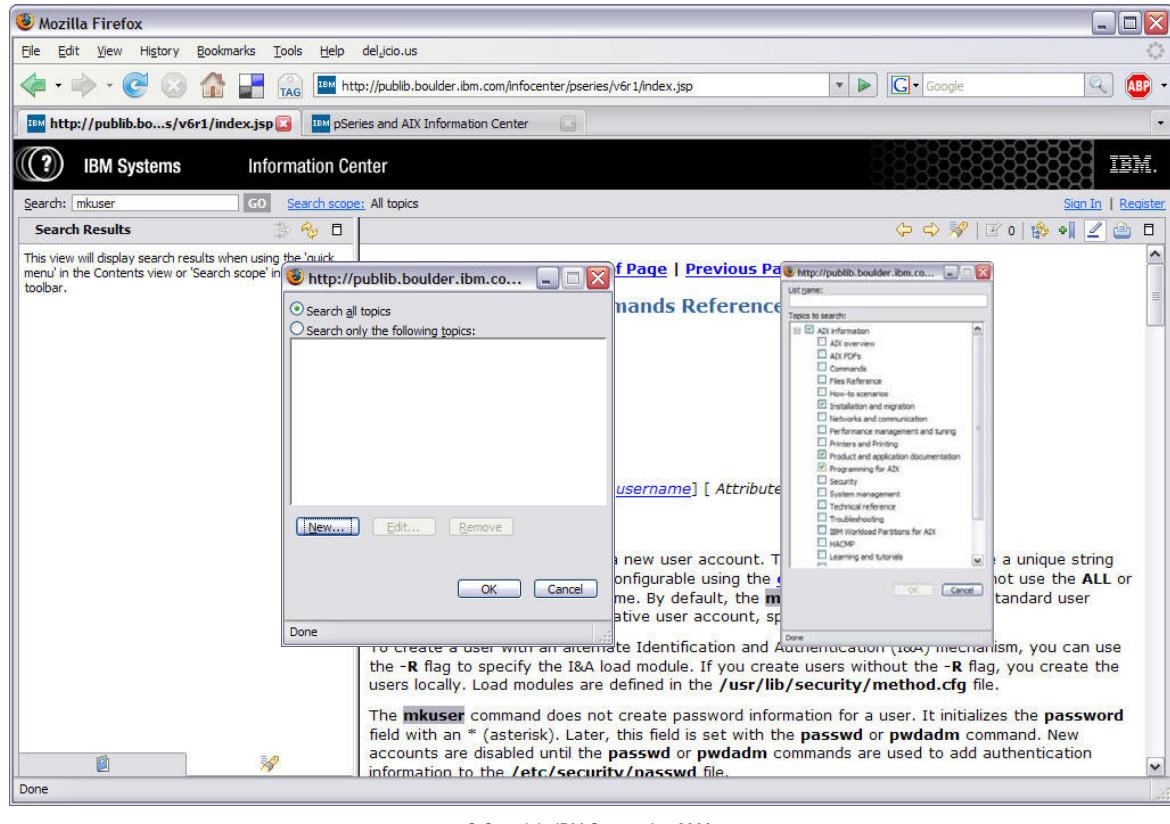


Figure 3-9. Search Scope

AU1310.0

Notes:

Search Scope

By selecting the **Search Scope** link, you can narrow the search to a subset of documents.

Instructor Notes:

Purpose — To show the search scope feature.

Details — Describe how to narrow down the search scope.

Additional Information — None.

Transition Statement — Let's test what you have learned in this unit with a few checkpoint questions.

Checkpoint

1. Which command displays manual entries online?

2. Complete the following sentences:

The AIX 6.1 online documentation is loaded on a _____ . Any other computer in the network with appropriate Web browser software can then become a _____ .

3. How can you start the documentation from the command line?

© Copyright IBM Corporation 2008

Figure 3-10. Checkpoint

AU1310.0

Notes:

Instructor Notes:

Purpose — To let the students test their knowledge on the covered material.

Details —

Checkpoint Solutions

1. Which command displays manual entries online? **man**
2. Complete the following sentences:
The AIX 6.1 online documentation is loaded on a **document server**. Any other computer in the network with appropriate Web browser software can then become a **document client**.
3. How can you start the documentation from the command line? **infocenter**

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — Let's test what you have learned in a machine exercise.

Exercise: AIX 6.1 Documentation



© Copyright IBM Corporation 2008

Figure 3-11. Exercise: AIX 6.1 Documentation

AU1310.0

Notes:

After completing the lab exercise, you will be able to:

- Execute the `man` command
- Initiate *Mozilla* to access AIX online documentation
- Use the AIX documentation

Instructor Notes:

Purpose — Transition to the lab.

Details — Tell the students what they will learn in the lab.

Additional Information —

Transition Statement — Let's wrap up this unit with a quick review of the key points we have covered.

Unit Summary

- The **man** command can be used from the command line to view descriptions of AIX commands.
- Use a **Web browser** to access **online documentation** with AIX 6.1.
- The online documentation and System p InfoCenter use the **same interface**.

© Copyright IBM Corporation 2008

Figure 3-12. Unit Summary

AU1310.0

Notes:

Instructor Notes:

Purpose — To wrap up the unit with a summary of this unit's key points.

Details —

Additional Information —

Transition Statement — Let's move on to a new unit.

Unit 4. Files and Directories

Estimated Time

01:15

What This Unit Is About

This unit introduces basic concepts for files and directories.

What You Should Be Able to Do

After completing this unit, students should be able to:

- Describe the different file types
- Describe the AIX file system structure
- Use both full and relative path names in a file specification
- Create, delete, and list directories
- Use the `touch` command to create an empty file

How You Will Check Your Progress

Accountability:

- Student Activity
- Checkpoint questions
- Exercise

Unit Objectives

After completing this unit, you should be able to:

- Describe the [different file types](#)
- Describe the AIX [file system structure](#)
- Use [full](#) and [relative path names](#) in a file specification
- Create, delete, and list [directories](#)
- Use the [touch](#) command to create an [empty file](#)

© Copyright IBM Corporation 2008

Figure 4-1. Unit Objectives

AU1310.0

Notes:

Instructor Notes:

Purpose — To introduce the unit objectives.

Details —

Additional Information —

Transition Statement — Let's begin by describing a file.

A File

- A file is:
 - A **collection of data**
 - A **stream of characters** or a "byte stream"
 - **No structure** is imposed on a file by the operating system

© Copyright IBM Corporation 2008

Figure 4-2. A File

AU1310.0

Notes:

Introduction

AIX imposes no internal structure on a file's content. The user is free to structure and interpret the contents of a file in whatever way is appropriate.

Instructor Notes:

Purpose — This visual describes the concept of a file to the students.

Details — To help clarify the points made here about who cares about the contents or structure of a file, try using as an example a spreadsheet or database file; only the spreadsheet or database tool understands the contents of the file.

Additional Information —

Transition Statement — Having defined what a file is, we now look at the different file types.

File Types

- **Ordinary:**

Text or code data

- **Directory:**

A **table of contents**, that stores a **list of files** within that directory

- **Special Files:**

Represent hardware or logical **devices**

Example: The **CD-ROM** device is accessed via the **/dev/cd0** file.

© Copyright IBM Corporation 2008

Figure 4-3. File Types

AU1310.0

Notes:

Various file types

Most *ordinary* files are comprised of alphanumeric characters and punctuation coded in ASCII and can be viewed, edited, and printed using common tools.

Some ordinary files have data coded in a format that is unique to an application and special tools are needed. For example, the file **/etc/utmp**, which contains a list of currently logged in users, can only be read by the **who** command.

Other files contain code that can be executed by users on the computer. These files are known as *binary files*. These programs are treated by AIX as ordinary files.

Directories contain a list of files within that directory, and the information the system needs to access the file data. Each directory entry represents either a file or another directory, known as a *subdirectory*.

Special files behave differently from ordinary files in that the data written to the file is processed by a device driver and typically sent to that device, such as a printer, rather than being stored in the file.

Instructor Notes:

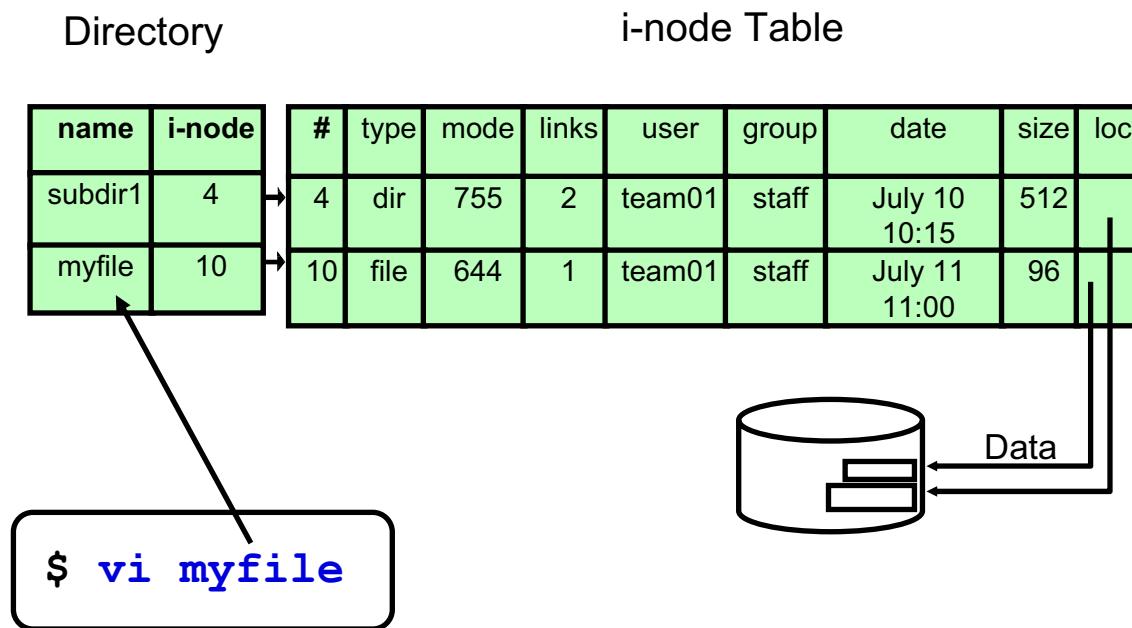
Purpose — To show the different file types that the system knows about.

Details — Remember to point out that, to AIX, EVERYTHING is a file! This will be important later when we cover redirection.

Additional Information — To try and help the student understand the difference between an ordinary file and a directory, use the analogy that a directory is like a filing cabinet, which is used to hold individual files. Filing cabinets are used as a method of storing files.

Transition Statement — All the information about a file is stored in the i-node for that file. We will now look at this concept.

Directory Contents



© Copyright IBM Corporation 2008

Figure 4-4. Directory Contents

AU1310.0

Notes:

Introduction

Directories enable you to group together related files and directories. A directory is a unique type of file that only contains enough information to relate a file name to the i-node which anchors and describes the file. As a result, directories usually occupy less space than ordinary files.

Directory contents

Each directory entry contains a file or subdirectory name and its associated index node (or i-node) number.

User access to files

When a user executes a command to access a file, they will use the file name. The system then matches the file name with the corresponding i-node number. Once the

i-node number is known, the system will access an i-node table, which holds information about the characteristics of the file.

i-node information

Examples of what is stored in the i-node table include the user ID of the owner of the file, the type of file, the date the file was last accessed and last modified, the size of the file, and the location of the file. Once the system knows the location of the file, the actual data can be located.

Instructor Notes:

Purpose — Describe the contents of a directory and the purpose of the i-node table.

Details — A directory is like a table of contents. It lists the names of files and subdirectories and their corresponding i-node number. When AIX wants to access a file or subdirectory, it uses the directory to find the i-node number, uses the i-node number to find the actual i-node entry, and finally uses the i-node's location information to find the file. The i-node entry in the i-node table contains all the important information about the file.

Be sure to point out that the ONLY place where the file name is stored is in the directory. File names are not stored in the i-node table. Also, do not get drawn into WHERE the i-node table is stored. This would lead to a discussion of file systems, which is beyond the scope of this class. From a user's perspective, they do not need to know the location of the i-node table.

At this time, do not go into detail on some of the i-node fields such as number of links and permissions. These will be covered in more detail later.

Explain to the students that a good understanding of the i-node structure will help them better understand the file and directory commands that will be discussed later in the topics (such as `ls`, `mv`, `cp`, and so forth).

Discussion Question — Why not just store the i-node information in the directory?

Answer: Keeping just name and i-node number information in the directory keeps directory management minimal and allows for more efficient use of disk space.

Additional Information —

Transition Statement — Having described how data is stored, let us now see what files and standard directories exist in AIX, as well as how this is referenced.

AIX File Systems

- Refers to both the physical and logical storage and access of files.
- In AIX, a **file system** is an **allocation of storage**.
- Similar in concept to partitions in the PC environment.
- Allows the operating system to store and retrieve the data from files quickly and efficiently.
- To access file systems, we associate them with a **directory**.
- AIX has several pre-defined file systems:

/ (root)	/tmp
/usr	/opt
/var	/home
/proc	

© Copyright IBM Corporation 2008

Figure 4-5. AIX File Systems

AU1310.0

Notes:

Physical file systems

The term *file system* refers to both the physical storage of the file data, and the logical organization of directories to allow quick and easy storage and retrieval of that data.

A physical file system is an allocation of storage, on a device such as a hard disk or a CD-ROM drive. The concept is very similar to *partitions* on some operating systems.

Space is assigned on a device, and associated with an identifier that the user can access. In AIX, this identifier is a directory in the system-wide *logical filesystem* structure.

If a physical file system fills up, then no more files can be created within the system.

The slide lists the pre-defined AIX physical file systems.

Instructor Notes:

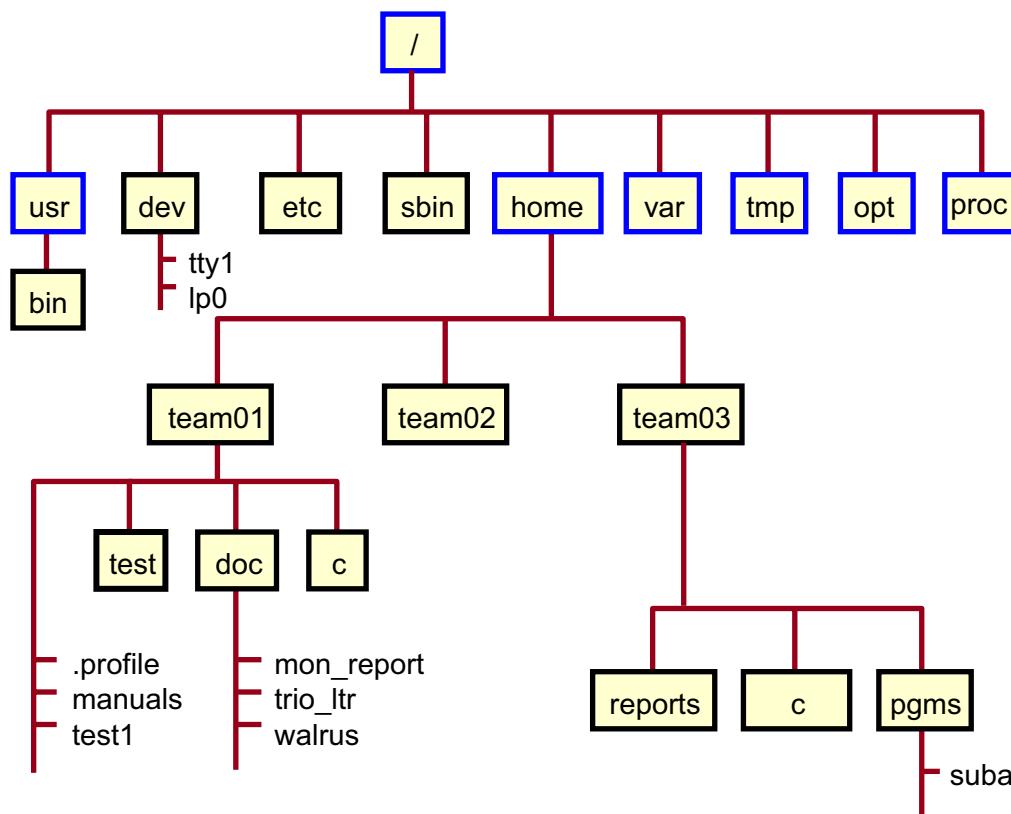
Purpose — Define what a the physical part of a file system is.

Details —

Additional Information — Do not go into too much detail here; we only want to introduce the physical aspect of file systems. This will be further defined in the AU14 (System Administration I) course.

Transition Statement —Let's now look at how these physical allocations are tied into directory hierarchy.

Hierarchical Structure



© Copyright IBM Corporation 2008

Figure 4-6. Hierarchical Structure

AU1310.0

Notes:

AIX file system

This file structure represents only part of a typical AIX file system. The file structure will always start at the **/ (root)** directory. It contains many directories that are critical in the operations of the system.

Examples

Some of the typical directories that can be found in an AIX **root** directory are:

- /sbin** - Utilities for system startup
- /dev** - Special files that represent devices
- /etc** - System configuration files

The **/usr** directory contains programs, libraries, and data files for various AIX applications:

The **/home** directory contains user login directories and files.

The **/var** directory contains files that dynamically change.

The **/tmp** directory will hold files created by executing programs.

The **/opt** directory is where third-party application programs can be stored. An example of files in this directory are the basic Linux commands, such as **tar**, **gzip**, **gunzip**, **bzip2**, and so forth, which are installed in the **/opt/freeware/bin** directory.

The **/proc** directory is where AIX maps processes and kernel data structures to corresponding files.

Accessing directories on other systems

It is also possible to access files on another computer in the network. Several facilities are available to do this, most notably, the Network File System (NFS). From a user's perspective, remote files will appear to behave just like local files.

Instructor Notes:

Purpose — To explain the hierarchical structure of AIX and the directories in which users will be working.

Details — Explain that the illustration will be referred to in the examples used in the course notes.

Point out that in this course, directories will always be denoted by words in boxes, whereas ordinary files will be words beside a line.

The four well known directories of **/home**, **/usr**, **/tmp** and **/var** allow their respective files to be accessed by users on the local system or, in some cases, on remote systems.

The contents of the **/usr** directory is for commands and utilities. Users do not have write access to this directory. Since these files are usually system-related, they do not change very much.

The **/home** directory is for user directories and files. Each user has a subdirectory whose name is the same as the login name. Users change their files regularly, therefore, this directory is very dynamic.

The **/tmp** directory is used by applications as a work space. For example, **vi** uses this area as a buffer space until the file is written to disk. Compilers will write files that are used during a compilation and when the compiler is finished the files are removed.

The **/var** directory is used for many user-type programs such as mail and printing. The name of the directory is short for variable since the data in this directory dynamically changes due to system activity.

The **/opt** directory is used with the basic Linux commands, such as **tar**, **gzip**, **gunzip**, **bzip2**, and so forth, which are installed in the **/opt/freeware/bin** directory.

The **/proc** directory is supported with AIX 5L V5.3. This pseudo file system maps processes and kernel data structures to corresponding files.

Point out that each user has a **home** directory allocated to them when the user is added to the system. Since it is their directory, they can do whatever they like in that directory, such as make subdirectories, create and delete files, and so forth. This is the directory they will be placed in when they log in.

Also mention that the top of the tree is denoted by the root directory, which is always referred to as **/**.

The student notes mention the Network File System (NFS). NFS is a TCP/IP application that is shipped with the base AIX operating system. Do not discuss NFS in any detail in this class. More advanced courses discuss NFS in further detail.

Additional Information — If there is interest, explain the AIX file tree in greater detail, such as the purpose of directories like **/etc** and **/usr**.

This will depend on the ability of the audience as well as their interest in the subject.

Transition Statement — We have talked about directories and the location of files, we will now discuss how to reference directories and files from the shell.

Path Names

- A **sequence of names**, separated by slashes (/), that describes the **path** the system must **follow** to **locate** a file in the file system.
- There are two types of path names:

Absolute or Full Path Name (start from the / directory):

```
$ vi /home/team01/doc/mon_report  
$ /usr/bin/ls -l /home/team01
```

- **Relative Path Name (start from current directory):**

```
$ cd /home/team01  
$ vi doc/mon_report  
$ cd /usr/bin  
$ ./ls -l /home/team01
```

© Copyright IBM Corporation 2008

Figure 4-7. Path Names

AU1310.0

Notes:

File path names

The path name is written as a string of names separated by forward slashes (/). The rightmost name is the file you wish to access. The other names are the directories and subdirectories that describe how to get to the file's location.

A path name is always considered to be **relative** unless it begins with a slash. An **absolute** or **full** path name always starts with a slash.

Instructor Notes:

Purpose — To explain the concept of paths to a file.

Details — You may want to draw the directory structure from Figure 4-6 on a free panel of the white board as it will be used in the examples.

Emphasize the direction of the slash!

Also ensure that you emphasize that in a path like **/home/team01/test** the first / represents the top of the tree structure that is the **root** directory, however, subsequent slashes used in the path are used as directory separators.

Additional Information — There are two special files in all directories:

- representing the current directory
- .. representing the directory above, or the parent directory to the current directory

Transition Statement — How would we know what the current directory is, when we are using relative path?

Where Am I?

- The **print working directory** command can be used to find out what your current directory is:

```
$ pwd  
/home/team01
```

© Copyright IBM Corporation 2008

Figure 4-8. Where Am I?

AU1310.0

Notes:

Using the **pwd** command

The **pwd** command will always return the full path name of your (current) present working directory. It is not a bad idea to use this command often, especially when you are removing files (to be sure that you are removing them from the correct directory).

Instructor Notes:

Purpose — To introduce the command that will tell us where we are sitting in the directory structure.

Details — Explain the usefulness of the `pwd` command by pointing out that at the moment we have no way of telling which directory we are in unless the `pwd` command is executed.

Later on the students will see how to add this type of information to their prompt.

Discussion Items — Ask the following:

If you were in `pgms`, what would be the result of typing in the `pwd` command?

Answer: `/home/team03/pgms`

Additional Information —

Transition Statement — Having now located where you are in the file tree, let's find out how to list the content of a directory.

Listing Directories

Syntax: **ls** [directory]

Common options:

- a: Show hidden files (files that start with a ".")
- R: List files in all subdirectories (recursively)

```
$ ls
c doc manuals test1

$ ls -a
. . . .profile c doc manuals test1

$ ls -R
c doc manuals test1

./c:
./doc:
Mon_report trio_ltr walrus
```

© Copyright IBM Corporation 2008

Figure 4-9. Listing Directories

AU1310.0

Notes:

Executing the **ls** command

The **ls** command is used to list the contents of a directory, and has many useful options with it. If no file or directory name is specified as an argument to the **ls** command, the current directory will be used.

By default, the **ls** command displays the information in alphabetic order. When the **ls** command is executed it does not display any file names that begin with a dot (.), unless the **-a** option is used (as can be seen on the visual). These files are generally referred to as hidden files, for this reason.

To list all the subdirectories and their contents, the **-R** option can be used. This is also known as a *recursive* directory listing.

Instructor Notes:

Purpose — This visual shows how the contents of a directory can be displayed and that there are differences in file names.

Details — Note the difference in the listing produced by the `ls` command and the listing produced by the `ls -a` command.

Transition Statement — There are many options that can be used with the `ls` command. Let's look at some of these options.

Long Listing of Files

The **ls** command with the **-l** option can be used to obtain more information about the files in a directory.

```
$ ls -l
total 5
drwxrwxr-x 2 team01 staff 1024 Aug 12 10:16 c
drwxrwxr-x 2 team01 staff 512 Feb 18 09:55 doc
-rwxrwxr-x 1 team01 staff 320 Feb 22 07:30 suba
-rwxrwxr-x 2 team01 staff 144 Feb 22 16:30 test1

$ ls -li test1
29 -rwxrwxr-x 2 team01 staff 144 Feb 22 16:30 test1
```

© Copyright IBM Corporation 2008

Figure 4-10. Long Listing of Files

AU1310.0

Notes:

File listing details

The fields from the **ls -l** command are as follows:

Permission bits	Link	Owner	Group	Size	Mtime	Name
drwxrwxr-x	2	team01	staff	1024	Aug 12 10:16	c
drwxrwxr-x	2	team01	staff	512	Feb 18 09:55	doc
-rwxrwxr-x	1	team01	staff	320	Feb 22 07:30	suba
-rwxrwxr-x	2	team01	staff	144	Feb 22 16:30	test1

Field 1 shows the file type (such as ordinary or directory) and the permission bits. File and directory permissions will be covered in more detail in a later unit.

Field 2 is the link count. Links will be covered in more detail in the next unit.

Field 3 shows the user name of the person who owns the file.

Field 4 shows name of the group for which group access privileges are in effect.

Field 5 shows the character count of the entry.

Field 6 shows the date the contents of the file or directory were last modified.

Field 7 shows the name of the file/directory.

The **-i** option used with the **ls** command displays the i-node number in the first column.

The **ls** command is merely displaying file and directory information from the i-node table. Only the last column, the name, comes from the directory itself.

Note the size of the directories in the above example. Directory space is allocated in 512-byte increments and grows in 512-byte increments.

Instructor Notes:

Purpose — The visual shows the output obtained from the `ls -l` and `ls -li` commands.

Details — Explain each field of the output in the following manner:

- Field 1 Only explain the significance of the first character.
 d for directory
 - for ordinary files
 Also warn users that they might see other characters in this field as well.
 The rest of the permission bits will be discussed later in the course.
- Field 2 Shows the link count. For directories the link count indicates the number of subdirectories in that directory. Directories will always have a link count of at least two (for . and ..). The link count for files indicates the number of names given to a single file. Links will be covered in more detail in the next unit.
- Field 3 Shows the owner of the file; usually the person who has created the file.
- Field 4 Displays the group that the file belongs to. Do not discuss the idea of groups here other than to say that every user on the system must belong to a group. Groups are a method of organizing users usually based on their job and the files that they need to access. Explain that this will become clearer when the permissions of a file are considered. The default group that comes with the system for ordinary users is called staff, which is what is seen on the visual.
- Field 5 Shows the number of characters.
- Field 6 Displays the time that the contents of the file or directory were last modified.
- Field 7 Displays the file name - which for a user is probably the most important field.

Be sure to mention that the `ls -l` command displays file information from the i-node table. The exception to that is the last column, the “name”, which is retrieved from the directory itself.

Also point out the sizes of the directories. Directory space is allocated in 512-byte increments. As a result, looking at the output of the `ls` command does not provide a good indication of the number of files and subdirectories in a directory.

Another point is that the size of a directory can only increase. Deleting files does not make the directory allocation size smaller. AIX will reuse the empty entry for future entries in the directory.

Decreasing the directory size involves moving the files to a newer, smaller directory. This involves commands not yet discussed, so you may not want to bring this issue up at this time.

One more important point for the students is that from a user standpoint, knowing the i-node number is not really all that important. However, understanding the concepts of i-node information provides a better understanding of how the various file manipulation commands work (such as `mv`, `cp`, and so forth). Also, since many students will eventually be AIX system administrators, this is a concept they will need to understand.

Students often ask about the Total that is displayed with the `ls -l` command. This value refers to the number of 512-byte blocks allocated to the actual files within the directory.

Additional Information —

- Field 2 When explaining link counts, you could also mention that when the `rm` command is issued, the link count is decremented by 1, and only when the link count is equal to 0, will a file be removed.
- Field 6 There are other flags that can be used with the `-l` flag which will display other time attributes. For example, `-c` will display the time that the i-node was modified, `-u` which displays the last access time to the file.

Transition Statement — Let's next introduce the `cd` command to change directories.

Change Current Directory

Syntax: `cd [directory]`

Set the current working directory from `/home/team01` to
`/home/team01/doc`:

```
$ cd doc          relative path  
$ cd /home/team01/doc    full path
```

Set your working directory to your home directory:

```
$ cd
```

Set your working directory to the parent directory:

```
$ cd ..
```

© Copyright IBM Corporation 2008

Figure 4-11. Change Current Directory

AU1310.0

Notes:

Introduction

The `cd` command is used to change your current working directory.

Returning to the home directory

Using the `cd` command with nothing after it will automatically return you to your home directory. This is the directory into which you are usually placed when you log in.

Instructor Notes:

Purpose — This visual shows how the current directory can be changed.

Details — Go through the examples using the hierarchical tree structure from earlier on. Also point out that with any of the commands that operate on directories, you can either use full path names or relative path names (as illustrated by the diagram). When using relative path names you must be certain where you are in the tree structure. If you are unsure, use the `pwd` command to find out.

Discussion Items — Ask the following question:

What do you think `$ cd ../../` does?

Answer: Moves you up two directories.

Also ask the students when would they want to change directories.

Additional Information —

Transition Statement — Let's do an activity to review what you have just learned.

Activity: Q + A

1. How can you determine the **i-node number** of a file?
2. Where are the **names** and **i-node numbers** of files stored?
3. How can you determine your **current directory**?
4. How can you list **all files in a directory**, including **hidden files**?
5. Your current directory is **/usr/dt/bin**. What is the easiest way to change to your **home directory**?
6. Which file names are **relative** (check all that apply)?
 ../team03/dir1
 /tmp/file1
 /.profile
 ./.profile
7. Write down the **three different file types** that AIX knows:
 - 1.
 - 2.
 - 3.

© Copyright IBM Corporation 2008

Figure 4-12. Activity: Q + A

AU1310.0

Notes:

Take some time and answer the questions.

Your instructor will review the questions with you afterwards.

Instructor Notes:

Purpose — To test if the students have understood what has been presented so far.

Details — Give the students some time to answer the questions. Afterwards review the questions:

1. `ls -li`
2. In the directory
3. `pwd`
4. `ls -a`
5. `cd`
6. relative, absolute, absolute, relative
7. ordinary, directory, special files

Additional Information —

Transition Statement — Let's look at creating our own directory structure.

Creating Directories

Syntax: **mkdir directory**

To create the directory **test**, as a sub-directory of **/home/team01**:

```
$ mkdir /home/team01/test           full path name
```

(or)

```
$ cd /home/team01  
$ mkdir test                      relative path name
```

© Copyright IBM Corporation 2008

Figure 4-13. Creating Directories

AU1310.0

Notes:

mkdir command

The **mkdir** command creates one or more new directories specified by the **dir_name** parameter. Each new directory contains the standard entries **.** (dot) and **..** (dot dot).

The **-m** option can be used with the **mkdir** command to specify the directory being created with a particular set of permissions. Permissions will be covered in Unit 6.

Instructor Notes:

Purpose — To demonstrate how users can create their own directory structure.

Details — Point out to the students that they will not have the permission to create directories wherever they please as it all depends on the permissions given to them (which will be discussed later).

Discussion Items — Ask the following:

1. What happens when you issue `$ mkdir test`, and `test` already exists
 - a. As a directory, and
 - b. As a file?

In both instances the system will complain that the file already exists.

2. Why do users create or need to create directory structures?

Additional Information — The `mkdir` command can accept more than one space delimited filename argument, thus allowing one to create multiple directories with a single `mkdir` execution.

example: `$ mkdir ./mydir /home/team01/doc/newdir`

Transition Statement — Let's explain the `rmdir` command.

Removing Directories

Syntax: **rmdir directory**

Remove the directory **/home/team01/test**:

```
$ rmdir /home/team01/test
```



The directory must be empty!

```
$ rmdir doc  
rmdir: Directory doc is not empty.
```

© Copyright IBM Corporation 2008

Figure 4-14. Removing Directories

AU1310.0

Notes:

Removing directories

You get no message if the command is successful. It never hurts to follow a command such as this with an **ls**, which is discussed on the next page, to make sure that you have accomplished what you set out to do.

A directory is considered empty if it contains only the **.** and **..** entries.

Instructor Notes:

Purpose — This visual shows how directories can be removed.

Details — The way a directory structure is removed is to first empty out each directory, then remove the directory. Also, you cannot be in the directory that you are trying to remove. This makes sense because if the system did allow you to remove a directory you were in, what would you be left in after the directory had gone? Tell the students to always think of the following “You cannot remove the ground that you are standing on!”

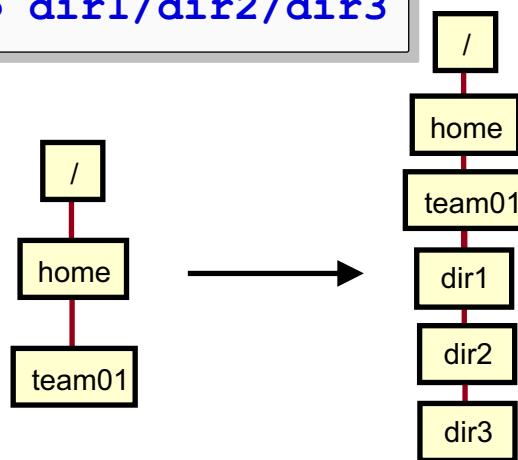
Additional Information — There are shortcuts, but these will be revealed when removing files is discussed.

Transition Statement — There is also a useful option that can be used with the `mkdir` and `rmdir` commands to work with multiple subdirectories.

Working with Multiple Directories

- Create multiple directories simultaneously:

```
$ mkdir -p dir1/dir2/dir3
```



- Remove all directories in the path specified:

```
$ rmdir -p dir1/dir2/dir3
```

© Copyright IBM Corporation 2008

Figure 4-15. Working with Multiple Directories

AU1310.0

Notes:

Multiple directories

Using the **-p** option with the **mkdir** command allows you to create multiple subdirectories simultaneously. If **dir1** and **dir2** already exist, then **dir3** will be created.

The **-p** option used with **rmdir** first removes the **dir3** directory, then the **dir2** directory, and finally the **dir1** directory. If a directory is not empty or you do not have write permission to it when it is removed, the command terminates with an error.

Instructor Notes:

Purpose — Explain how the `-p` option works with the `mkdir` and the `rmdir` commands.

Details — The `-p` option with the `mkdir` or `rmdir` commands allow you to create and remove multiple subdirectories simultaneously. Not all versions of UNIX allow this syntax.

The `-p` option can be considered to mean parent or path.

Transition Statement — Let's introduce the `stat` command to query i-nodes.

Displaying Directory Information

```
$ ls -ldi mydir
51 drwxr-xr-x 2 team01 staff 512 Jan 17 17:38 mydir

$ istrat mydir
Inode 51 on device 10/8 Directory
Protection: rwxr-xr-x
Owner: 208(team01) Group: 1 (staff)
Link count: 2 Length 512 bytes

Last updated: Thu Jan 17 21:05:43 2002
Last modified: Thu Jan 17 17:38:52 2002
Last accessed: Fri Jan 18 13:30:00 2002

$
```

© Copyright IBM Corporation 2008

Figure 4-16. Displaying Directory Information

AU1310.0

Notes:

Using the **istrat** command to query i-nodes

The **-i** option displays the i-node number in the first column. The **-d** option used with **ls** will list the i-node information for a directory.

The **ls** command has options that can display each of the timestamps:

- To display the updated time (utime): **ls -lc**
- To display the modification time (mtime): **ls -l**
- To display the access time (atime): **ls -lu**

istrat displays the i-node information for a particular file or directory. AIX systems maintain three timestamps for files and directories. The updated time reflects changes done to the i-node information; whereas, the modification time reflects changes to the contents of the file or directory itself. The access time is the last time the file was read or written. Reading a file changes its access time, but not its updated time or modification time, because information about the file or directory was not changed.

Instructor Notes:

Purpose — Describe two commands that will show you directory information.

Details — A directory is a file. That is why the `istat` and `ls` commands can be used to display the information about directories as well as files.

As you already know, the `ls` command displays the contents of each specified file parameter. Using `ls` without any parameters displays the contents of the current directory.

The `-i` option displays the i-node number in the first column of the report.

The `-d` option displays only the information for the directory. Directories are treated like files, which is helpful when using the `-l` option to get the status of a directory.

The `istat` output gives you additional information about the file or directory in the form of three timestamps. Updated and modification differ in that updated is like changing the wrapping paper on a gift; whereas, modification is like changing the gift that is inside the box. Useful boardwork to show the difference is: `chmod 755 mydir` is an update. Doing `echo date >> myfile` is a modification. Updated always refers to the i-node information. Modification refers to the contents of the file or directory. Access time changes any time the file or directory is read or written. The key here is “read.” If a user or an application reads the contents of the directory, the access time changes. Neither the updated nor modification times change.

Additional Information — Only **root** can execute the `istat` command using the i-node number and device rather than a file or directory name. The same information is displayed plus the hexadecimal block pointers. These numbers are addresses of the disk blocks that make up the file or directory.

The 10/8 displayed with `istat` represents the major number (10) associated with disks and the minor number (8) for `/dev/hd1`.

When using a directory name as an argument, one must use the `-d` flag to see the attributes of the directory; otherwise, the `ls` command will only list the attributes of the files underneath that directory.

Transition Statement — Now that you know about working with directories, let's look at the rules for naming files and directories.

AIX File Names

- Should be **descriptive** of the content
- Should use only **alphanumeric characters**:
 - **UPPERCASE**, lowercase, number, #, @, _
- Should not include imbedded **blanks**
- Should not contain **shell metacharacters**:
 - * ? > < / ; & ! [] | \$ \ ' " ()
- Should **not** begin with "+" or "-" sign
- Should **not** be the **same** as a **system command**
- Are **case-sensitive**
- File names starting with a **.** (dot) are hidden from the normal ls command
- The **maximum number of characters** for a file name is **255**

© Copyright IBM Corporation 2008

Figure 4-17. AIX File Names

AU1310.0

Notes:

AIX file names

AIX has no notion of file name extensions as you have in other operating systems (such as DOS). The dot is simply used as part of the file name. Applications may use this to describe the contents of a file, but it is not enforced or required by AIX.

Instructor Notes:

Purpose — To introduce to the students the rules of file names in AIX.

Details — Point out that reasonable length file names (up to 16 characters) are recommended.

Some applications can work only with 8+3 file names (such as DOS), usually for compatibility with other environments.

Some flavors of UNIX restrict the length of file names to only 14 characters.

Other applications apply their own suffixes to a file name to denote the file type (**.tmp**, **.sam**, ...)

Explain the use of the term *should* on the slide. It is possible to create file names which violate these guidelines, but if you do you will have difficulty using them as arguments to commands. This is due to special metacharacter meanings that we will be covering later in the course.

Additional Information —

Transition Statement — We are going to discuss one more command before the unit ends. This command is the **touch** command.

touch Command

- The **touch** command updates the **access** and **modification times** of a file. The command can also be used to **create zero-length files**.

```
$ ls -l  
-rwxrwxr-x    1  team01 staff    320  Jan  6 07:30 suba  
$ date  
Tues Sep 10 12:25:00 2002  
$ touch suba new_file  
$ ls -l  
-rwxrwxr-x    1  team01 staff    320  Sep 10 12:25 suba  
-rw-r--r--    1  team01 staff      0  Sep 10 12:25 new_file
```

© Copyright IBM Corporation 2008

Figure 4-18. **touch** Command

AU1310.0

Notes:

Creating empty files

The **touch** command serves two purposes. If the file specified by the file name does not exist, a zero-length (empty) file is created. If the file does exist, the last modification time (displayed with **ls -l**) is updated to reflect the current date and time.

If you do not specify a time variable with the **touch** command the current date and time will be used.

touch can also be helpful when used in situations where an application checks a file's last modification time before taking some action such as backup or compile.

Instructor Notes:

Purpose — To discuss the usefulness of the `touch` command.

Details — We are covering the `touch` command here as it will be used in the next machine exercise.

Explain the use of the `touch` command and go through the given example. Point out that with the `touch` command, if a file name is specified that does not exist, then a zero-length file will be created. If the `-c` option is used, the file will not be created if it does not already exist.

Additional Information — The time variable referred to is the `-t` option of the `touch` command. It accepts a value which is referred to as the “time variable.” An example of the `touch` command using a time variable is:

```
$ touch -t 02171425 myfile
```

In this case the time stamp will be 14:25 (2:25 p.m.) February 17 of the current year.

Transition Statement — Before moving to the machine exercise, let's review some checkpoint questions.

Checkpoint (1 of 2)

1. Using the tree structure shown earlier, and using **/home** as your current directory, how would you refer to the **suba** file in the **pgms** directory using both full and relative path names?

2. When specifying a path name, what is the difference between the **.** and the **..**?

3. What will the **cd . . / . .** command do?

4. What conditions have to be satisfied in order for the **rmdir** command to complete successfully?

© Copyright IBM Corporation 2008

Figure 4-19. Checkpoint (1 of 2)

AU1310.0

Notes:

Instructor Notes:

Purpose — To test the students' knowledge of this unit's materials.

Details —

Checkpoint Solutions (1 of 2)

1. Using the tree structure shown earlier, and using **/home** as your current directory, how would you refer to the **suba** file in the **pgms** directory using both full and relative path names?

Relative path name: **team03/pgms/suba**
 Full path name: **/home/team03/pgms/suba**

2. When specifying a path name, what is the difference between the **.** and the **..**?
 - . **Specifies current directory**
 - .. **Specifies parent directory**
3. What will the **cd . . / . .** command do?
Change your current working directory two directories higher.
4. What conditions have to be satisfied in order for the **rmdir** command to complete successfully?
 1. **The directory must be empty.**
 2. **You must be at least one directory level higher than the one you are trying to remove.**

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — A few more questions...

Checkpoint (2 of 2)

5. Match the various options of the ls command with their functions.

-a	____ Provides a long listing of files
-i	____ Lists hidden files
-d	____ Lists subdirectories and their contents recursively
-l	____ Displays the i-node number
-R	____ Displays information about a directory

6. Circle the following valid file names in the following list:

1
aBcDe
-myfile
my_file
my.file
my_file
.myfile

© Copyright IBM Corporation 2008

Figure 4-20. Checkpoint (2 of 2)

AU1310.0

Notes:

Instructor Notes:

Purpose — To test the students' knowledge of this unit's materials.

Details —

Checkpoint Solutions (2 of 2)

5. Match the various options of the ls command with their functions.

-a	<u>-l</u>	Provides a long listing of files
-i	<u>-a</u>	Lists hidden files
-d	<u>-R</u>	Lists subdirectories and their contents recursively
-l	<u>-i</u>	Displays the i-node number
-R	<u>-d</u>	Displays information about a directory

6. Circle the following valid file names in the following list:

- 1
aBcDe
- myfile
- my_file
- my.file
- my_file
- .myfile

© Copyright IBM Corporation 2008

Additional Information — The term "valid" in question 6 is to be read as meaning a file names which will not cause confusion to the shell or to a command using it as an argument.

Transition Statement — Let's switch to the lab exercise.

Exercise: Files and Directories



© Copyright IBM Corporation 2008

Figure 4-21. Exercise: Files and Directories

AU1310.0

Notes:

After this exercise, you will be able to:

- Work with directories
- Use the `ls` command
- Use the `touch` command

Instructor Notes:

Purpose — Introduce the lab.

Details — Provide what students will learn in the lab.

Additional Information —

Transition Statement — Let's summarize the key points from this unit.

Unit Summary

- There are three types of files which are supported:
 - Ordinary
 - Directory
 - Special
- The AIX file system structure is a hierarchical tree.
- Files are accessed using either full or relative path names. A path name always begins with a / (forward slash).
- The following commands can be used with directories:
pwd, cd, mkdir, rmdir, and ls.

© Copyright IBM Corporation 2008

Figure 4-22. Unit Summary

AU1310.0

Notes:

Instructor Notes:

Purpose — Describe the key points from this unit's material.

Details —

Additional Information —

Transition Statement — Let's move on to another unit.

Unit 5. Using Files

Estimated Time

00:40

What This Unit Is About

This unit introduces useful commands to be used when working with AIX files.

What You Should Be Able to Do

After completing this unit, you should be able to:

- Use the `cp` command to copy files
- Use the `mv` command to move or rename files
- Use the `wc` command to count the number of lines, words, and bytes in a named file
- Use the `ln` command to allow a file to have more than one name
- Display the contents of a file using the `cat`, `pg`, and `more` commands
- Use the `rm` command to remove files
- Print files

How You Will Check Your Progress

Accountability:

- Student activity
- Checkpoint questions
- Exercise

Unit Objectives

After completing this unit, you should be able to:

- Use the **cp** command to copy files
- Use the **mv** command to move or rename files
- Use the **wc** command to count the number of lines, words, and bytes in a named file
- Use the **ln** command to allow a file to have more than one name
- Display the contents of a file using the **cat**, **pg**, and **more** commands
- Use the **rm** command to remove files
- Print files

© Copyright IBM Corporation 2008

Figure 5-1. Unit Objectives

AU1310.0

Notes:

Instructor Notes:

Purpose — To introduce the unit objectives.

Details —

Additional Information —

Transition Statement — Let's start by looking at the file copy process.

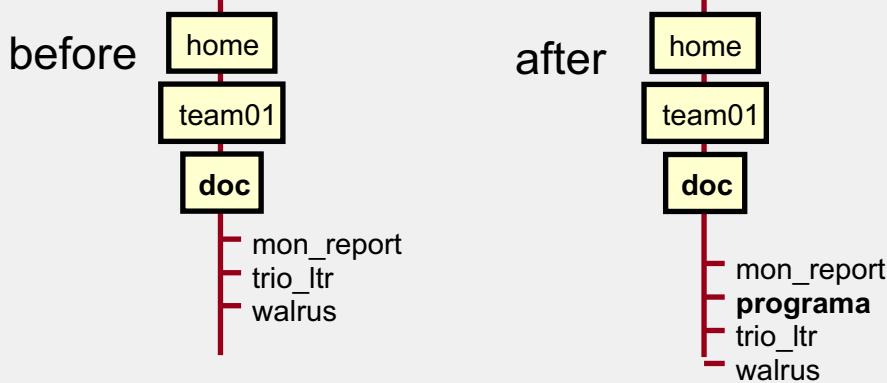
Copying Files

```
cp source target  
cp file1 file2 ... target_dir
```

To copy the file `/home/team03/pgms/suba` to `/home/team01/doc` and name it **programa**:

```
$ pwd  
/home/team01/doc
```

```
$ cp /home/team03/pgms/suba programa
```



© Copyright IBM Corporation 2008

Figure 5-2. Copying Files

AU1310.0

Notes:

Copying files

The `cp` command copies from one or more source files to a target file. The source is usually an ordinary file, but can also be a directory file if the `-R` (recursive) option is used.

Instructor Notes:

Purpose — To introduce the `cp` command.

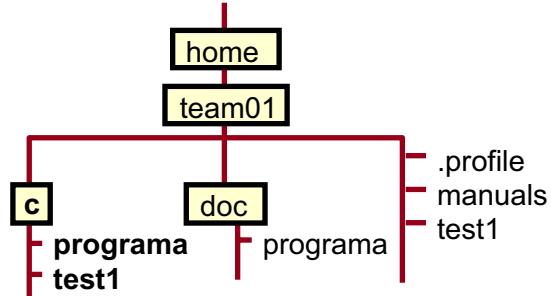
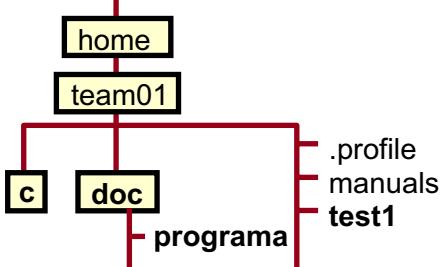
Details — Go over each of the `cp` examples discussing what is happening. For clarification, the two file structures represent before and after images for the `/home/team01/doc` directory.

Additional Information — Note that the file **programa** now holds the same data as is in the file **suba**. And that the file **suba** is unchanged (other than the time accessed timestamp).

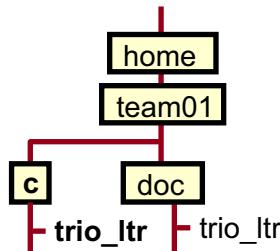
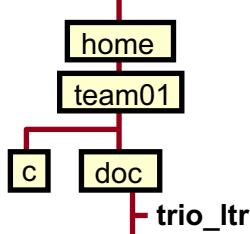
Transition Statement — Let's look at some examples.

cp Examples

```
$ cd /home/team01
$ cp doc/programa test1 c
```



```
$ cd /home/team01/doc
$ cp trio_ltr ../c
```



© Copyright IBM Corporation 2008

Figure 5-3. cp Examples

AU1310.0

Notes:

Copying multiple files

The `cp` command can copy multiple files to a target. In that case, the target must be a directory.

Target exists

When using the `cp` command, if the file specified as the target file already exists, then the copy operation will overwrite the original contents of the file without warning. To avoid this use the `-i` (interactive copy) option.

Target is a directory

If the target is a directory, the source file(s) will be created within the directory, and retain the same file name(s).

Recursive copy

`cp -R` can be used to recursively copy all files, subdirectories, and the files in those subdirectories to a new directory. For example:

```
cp -R /home/team01/mydir /home/team01/newdir
```

Question:

What command would you use to copy the file **/public/phonebook** into your current directory? (Hint: You do not need to know what your current directory is.)

Instructor Notes:

Purpose — These visuals show how files can be copied from one directory to another.

Details — Remember to be very clear about one point: The syntax of the command is `cp SOURCE TARGET`. The most common problem at this stage is students forgetting the target portion of the command.

Go through each example, as each one illustrates a slight variation of the command. Two diagrams are given with each example which should be used as a before and after picture.

At this point, the students often have as much trouble following the relative path usage as they do with the `cp` command syntax. You may wish to use this as an opportunity to test, review, and reinforce the relative path concepts.

Additional Information — The `cp` command allows recursive copying of data contained in the source to the target, thus allowing the replication of complete data trees. Use `cp -R` to accomplish this.

Discussion Items — In answer to the question in the student notes, use the command:

```
cp /public/phonebook .
```

Ask the following question:

What affect do you think the `cp` command would have on i-node information?

Answer: The source file would remain the same (although its last access time would be updated). A new i-node number and entry would be created for the newly copied (target) file.

Transition Statement — What if we wish to move a file, rather than copy it to another directory? Let's see how this can be achieved.

Moving and Renaming Files

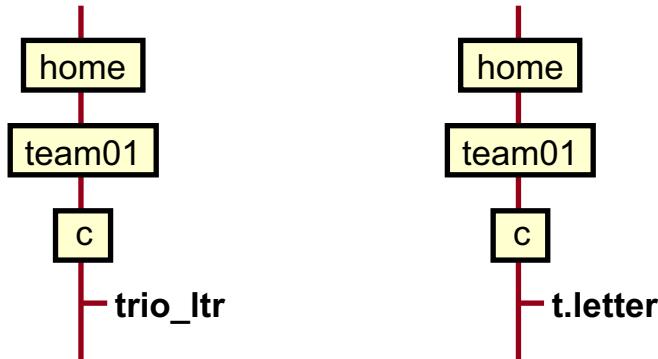
```
mv source target  

mv file1 file2 ... target_dir
```

```
$ pwd  

/home/team01/c  

$ mv trio_ltr t.letter
```



© Copyright IBM Corporation 2008

Figure 5-4. Moving and Renaming Files

AU1310.0

Notes:

The **mv** command

The **mv** command moves files and directories from one directory to another or renames a file or directory. If you move a file or directory to a new directory, it retains the base file name. When you move a file, all links to other files remain intact, except when you move it to a different file system. When you move a directory into an existing directory, the directory and its contents are added under the existing directory.

When you use the **mv** command to rename a file or directory, the target can specify either a new file name or a new directory path name.

Instructor Notes:

Purpose — To show how to rename or move a file.

Details — Review the example with students. Several more examples are shown on the next visual.

Discussion Item — Ask the students:

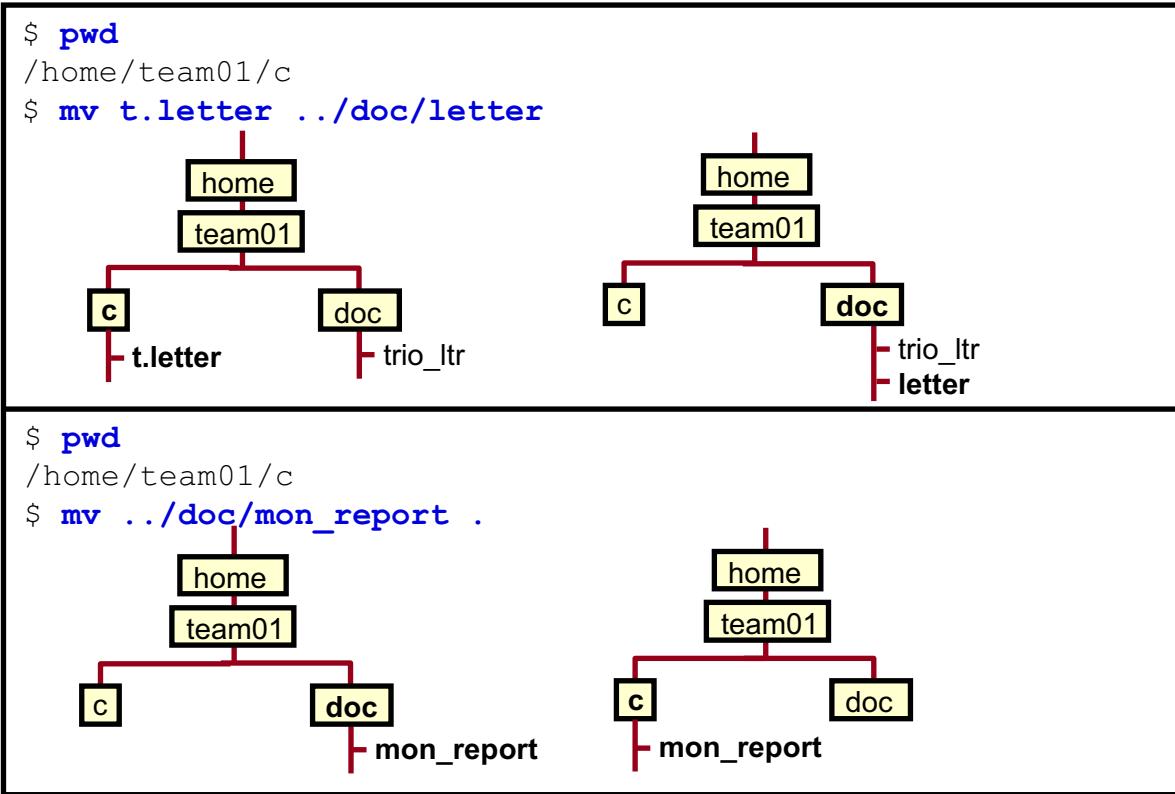
Question: What affect would the `mv` command in this example have on the i-node information?

Answer: The i-node number would remain the same. Only the last updated entry would be changed because the name of the file has changed (our examples assume that we are working within the same file system).

Additional Information — This is an excellent opportunity to build on top of the directory and i-node concepts taught earlier. Draw a diagram(s) of a directory file(s) with filename entries pointing to i-nodes which anchor the data files. Use the diagram(s) to show a move between directories (create a new entry in the destination and remove the entry in the source directory). Emphasize that in most cases the i-nodes and data file do not physically move. We are just changing the directory file entries. Then show the same operation where we are only changing the file name under a single directory. This helps to explain why a rename (in most instances) is just a rename and not an actual move of the file contents.

Transition Statement — We saw how to rename a file. Let's now look at other actions you can use with `mv`.

mv Examples



© Copyright IBM Corporation 2008

Figure 5-5. mv Examples

AU1310.0

Notes:

Moving files

As a result of the `mv` command, you will still have the same number of files as you did before. Furthermore, all the attributes remain the same. The only things that change are the file name and location.

Command arguments

The source can be a file or a list of files. If the source is a list of files, then the target must be a directory.

The target can be a file or a directory. **Warning!** If the target is the name of a file that already exists and if you have the correct permissions set for that file and directory, you will overwrite the file and never get an error message. To avoid this, use `mv -i`, an interactive move which prompts you if there are duplicate names.

Instructor Notes:

Purpose — To show how files can be moved from one directory to another, or simply renamed.

Details — Again the diagrams are intended to be used in their pairs to illustrate a before and after situation. Ensure all examples are explained as they each highlight a slight variation to the base scenario.

Additional Information — The `mv` command can overwrite many existing files unless the `-i` (interactive) flag is used.

All `mv` operations within a file system are only renames (the full path to a file is the full name of that file). When we do an `mv` between file systems, the operation is effectively a `cp` of the data to the target filesystems (with the allocation of a new i-node), followed by the removal of the i-node and the data file in the source file system. Thus moving large sized files or large amounts of files seem almost instantaneous when done within the same file system while the same operation between file systems can take a long time.

Transition Statement — Let's introduce the `cat` command.

Listing File Contents

```
cat file1 file2 ...
```

```
$ cat walrus
```

"The time has come," the Walrus said,
"To talk of many things:
Of shoes - and ships - and sealing wax -
Of cabbages - and kings -
And why the sea is boiling hot -
And whether pigs have wings."

From The Walrus And The Carpenter
by Lewis Carroll (1871)

© Copyright IBM Corporation 2008

Figure 5-6. Listing File Contents

AU1310.0

Notes:

Introduction

The **cat** command displays the contents of all the files that are specified as arguments to the command.

Too much output?

The **cat** command does not paginate the output. If it is longer than a screen, the file will scroll until the bottom of the file is reached. Thus, you may only be able to read the last full screen of information.

Line numbers

To display all the lines of a file, with numbers displayed beside each, use the **-n** flag.

Instructor Notes:

Purpose — To illustrate one command which can be used to view the contents of the file.

Details — If you do not specify a file name with the `cat` command, the command reads from standard in. Standard out is the default with the `cat` command output. Be careful about standard in and out here as they will be covered later in the course.

The command can also be used to concatenate the contents of many files. We will see later in the course how to create a file by combining the `cat` command with the redirect output symbol.

Discussion Items — Ask the students if they can see a problem with the `cat` command. The obvious answer is no pagination.

Ask the students, based on their current knowledge, how they might stop and start the scrolling of the output from the `cat` command?

Answer: Use `<Ctrl-s>` to freeze the screen and `<Ctrl-q>` to start the scrolling.

Additional Information —

Transition Statement — We obviously need to consider commands which will paginate the output. Let's see how pagination can be achieved.

Displaying Files

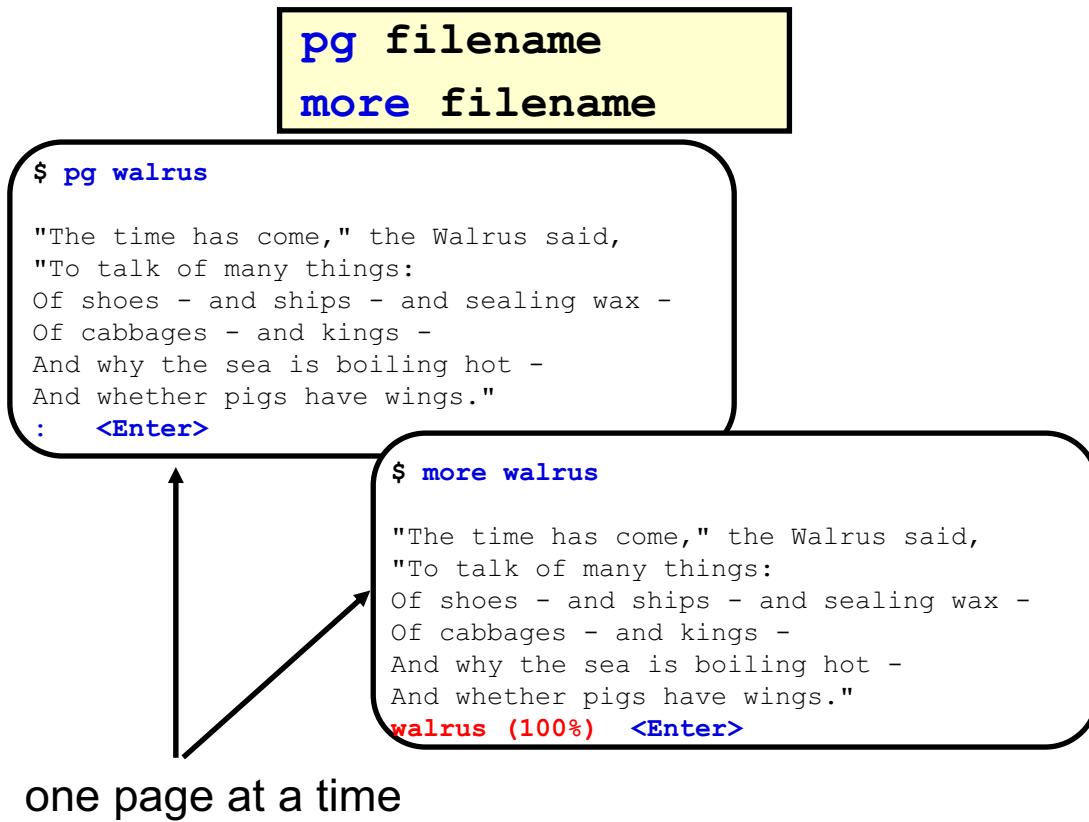


Figure 5-7. Displaying Files

AU1310.0

Notes:

Displaying files

The **pg** command reads the file names specified and displays the files one page at a time. Each screen is followed by a prompt. Press **<Enter>** to display the next page down and **h** to get help information.

The **more** command works in much the same way as the **pg** command; it displays continuous text one screen at a time. It pauses after each screen and prints the word *More* at the bottom of the screen. If you press Enter, it displays an additional line. If you press the **<space bar>**, it displays the next screen of text.

Search function

Both **pg** and **more** support searching through the file for patterns. **more** also supports search pattern highlighting.

Instructor Notes:

Purpose — To illustrate how to display the contents of a file in a paginated manner.

Details — Explain the subtle differences of the two commands.

The `cat`, `pg`, and `more` commands are good examples of how there may be several ways to accomplish a single task in AIX. By the way, the `pg` command came from AT&T V5, while the `more` command came from BSD UNIX.

Also point out that the `man` pages use the `more` command (which the students should be familiar with).

Mention that with either command if users want to come out of the program back to the command line, they can use the `<Ctrl+c>` key sequence.

Additional Information — Apart from moving down one page at a time, both the `pg` and the `more` commands support subcommands to perform other functions, for example to move up by one page with `more` use the `b` character. To move up one page with the `pg` command use the `-` (dash) character. Do not forget that with the `pg` command you must press the Enter key after a subcommand, whereas with the `more` command the action takes place as soon as the subcommand is entered.

For further information see `man` or `info`.

As you will remember, AIX has no knowledge of the structure of the data in a file; it is just a stream of characters. It is the text processing applications such as the `wc` command (along with editors and display commands) which understand certain standard conventions in text files. In the case of counting “lines” the `wc` command counts the strings delimited by newline (`\n`) characters. On the other hand a “word” is a string delimited by white space (for example, spaces or tabs).

Transition Statement — Let's explain the `wc` command.

wc Command

The **wc** command counts the number of lines, words, and bytes in a named file:

```
$ wc [-c] [-l] [-w] filename
```

Options:

- c counts the number of bytes
- l counts lines
- w counts words

Example:

```
$ wc myfile
```

17	126	1085	myfile
			characters
			words
			lines

© Copyright IBM Corporation 2008

Figure 5-8. **wc** Command

AU1310.0

Notes:

Counting file contents

When files are specified with the **wc** command, their names will be printed along with the counts. If options are not used, the order of the output will always be lines, words, and characters.

Instructor Notes:

Purpose — To discuss the use of the `wc` command.

Details — Discuss the syntax and the output shown as in the example. Also define that a word is seen as a string of characters delimited by spaces, tabs, or the new line characters.

Additional Information —

Transition Statement — Let's do an activity.

Activity: Working with the wc Command



© Copyright IBM Corporation 2008

Figure 5-9. Activity: Working with the wc Command

AU1310.0

Notes:

Activity

- ___ 1. Log in to the system with your **teamxx** ID and password.
- ___ 2. Execute the **wc** command and count lines in file **.profile**.
- ___ 3. Execute the **wc** command and count the bytes in file **.profile**.
- ___ 4. Execute the **ls -la** command on the file **.profile**. Compare this number with the output of in the previous step.

Activity with Hints

- ___ 1. Log in to the system with your **teamxx** ID and password.
 - » login: **teamxx** (at the login prompt)
 - Password: **teamxx** (default password same as user name)
- ___ 2. Execute the **wc** command and count lines in file **.profile**.
 - » **\$ wc -l .profile**
- ___ 3. Execute the **wc** command and count the bytes in file **.profile**.
 - » **\$ wc -c .profile**
- ___ 4. Execute the **ls -la** command on the file **.profile**. Compare this number with the output of in the previous step.
 - » **\$ ls -la .profile**

Instructor Notes:

Purpose — Processing phase.

Details — Give the students some time to work on this activity.

Additional Information —

Transition Statement — Let's introduce the `ln` command.

Linking Files

```
ln source_file target_file
```

- Allows files to have more than one name in the directory structure
- Both files reference the same i-node
- Cannot be used with directories, cannot span file systems

```
$ ls -li
63 -rw-r--r--    2 team01    staff      1910 Nov 21 14:19 man_files

$ ln man_files manuals

$ ls -li
63 -rw-r--r--    2 team01    staff      1910 Nov 21 14:19 man_files
63 -rw-r--r--    2 team01    staff      1910 Nov 21 14:19 manuals

$
```

© Copyright IBM Corporation 2008

Figure 5-10. Linking Files

AU1310.0

Notes:

Introduction

The **ln** command in its simplest of forms allows one file to have two or more different names in the directory structure. These multiple references are also known as *hard links*. No copy of the file takes place; each reference in the directory points to the same i-node.

All other characteristics of the original file are reflected in the new file reference. That is, permissions, ownership, size, and modification time all remain the same. Any changes to either file will be reflected in the other.

When using the **ln** command, the source is the existing file, and the target is the new file reference (or link) to be created.

These types of file references can only be created within the same AIX file system (or partition).

Instructor Notes:

Purpose — To illustrate how the `ln` command can be used to have one copy of a file referenced by multiple names.

Details — Explain that in the example shown `/home/team01/manuals` is linked to a file called `/home/team02/man_files`. There is only one physical copy of the file, however the file can be referenced by two names.

Discussion Item — Ask the students what affect a link would have on the i-node number of a file. Answer: The i-node number for both copies is the same. This is how it is possible to use multiple names for one physical copy.

You may also want to point out the link count column; it specifies how many references a particular i-node has in the physical file system. Files normally have a link count of 1; directories always have a link count of 2 (because of the “.” and “..” file entries).

Additional Information — To find out more information about `ln` see the manual pages.

If a student does ask, the link we are describing here in class is known as a *hard link*. A hard link allows a file to have multiple names.

Transition Statement — Let’s take a look at the other type of reference that `ln` can create; a symbolic link.

Linking Files (cont.)

```
ln -s source_file target_file
```

- Creates an **indirect** reference to a file (**symbolic link**)
- Name references the original file's name and path
- Can be used with directories and span file systems

```
$ ls -li
63 -rw-r--r--    2 team01    staff     1910 Nov 21 14:19 man_files

$ ln -s man_files manuals

$ ls -li
63 -rw-r--r--    1 team01    staff     1910 Nov 21 14:19 man_files
66 lrwxrwxrwx    1 team01    staff     1910 Nov 21 14:19 manuals -> man_files

$
```

© Copyright IBM Corporation 2008

Figure 5-11. Linking Files (cont.)

AU1310.0

Notes:

Introduction:

The **ln** command also allows for the creation of indirect references to files. These are called *symbolic links*, and are created using the **-s** option.

When a symbolic link is created, the directory entry references the name of the source file, rather than the i-node. They can be used with directories as well as files, and can span physical file systems.

In the **ls -l** output above, the symbolic link is visible in the file permission bits; the first character is an **l** (lowercase l). Also, in the link file name, you can see the reference to the source file.

Instructor Notes:

Purpose — Discussed symbolic links

Details — You may also want to point out that in the output above, the column in the output that describes the link count, each file only has one “link.” This is in contrast to the previous slide on hard links, where the link count for both files incremented each time a new hard link is created.

Additional Information — None

Transition Statement — Next, let’s look at how files can be removed.

Removing Files

```
rm file1 file2 file3 ...
```

```
$ ls
mon_report    trio_ltr    walrus

$ rm mon_report
$ ls
trio_ltr    walrus

$ rm -i walrus
rm: remove walrus?y

$ ls
trio_ltr

$
```

© Copyright IBM Corporation 2008

Figure 5-12. Removing Files

AU1310.0

Notes:

Methods of removing entries

The **rm** command removes the entries for the specified file or files from a directory. By default, there is no confirmation before the entry is deleted. To include a confirmation, use the **-i** (interactive) option.

The **-r** option permits recursive removal of directories and their contents if a directory is specified. Use this option with care, as it does not require the directory to be empty in order for this option to work.

Instructor Notes:

Purpose — To demonstrate the different methods of removing files.

Details — Discuss each command, going through the examples, pointing out the dangers of the `rm` command. Mention that the `rm` command may ask for confirmation for the removal of the files specified. If specified files are successfully removed, no output is given to indicate this. For example, if using the `rm` command to remove a file you own that has write permission on it, there is no confirmation necessary. However, if your file does not have write permission, confirmation is necessary.

Additional Information — Although we have not yet discussed permissions, it is important to bring to the students' attention the file and directory permission requirements to remove a file. Be sure to refer to the `rm` command when discussing permissions.

The `del` command can be optionally installed on an AIX system. It is interactive by default and performs like the `rm -i` command.

Transition Statement — Let's see how we can send a file off to the printer to be printed.

Printing Files

- **qprt** - queue files to the printer
- **qchk** – display the current status of a print queue
- **qcan** – cancel a print job (specify job number)

```
$ qprt walrus

$ qchk
Queue   Dev Status   Job Files     User      PP%   Blks Cp Rnk
lp0     lp0 Running    99 walrus team01     1      1  1   1

$ qcan -x 99

$
```

© Copyright IBM Corporation 2008

Figure 5-13. Printing Files

AU1310.0

Notes:

Print subsystems

AIX supports two printing subsystems: the System V Printing Subsystem and the AIX Printing Subsystem.

Printing files

The printer queue mechanism of either subsystem allows multiple users to use the same printer without a user having to wait for the printer to be available.

To queue a file for printing there are a number of commands available (to remain compatible with other versions of UNIX). They are **qprt**, **lp**, **lpr**. The command **qprt** has the most facilities.

To specify a printer (other than the default) use the **-P** option to the **qprt** command. For example, to send a file to queue **lp1** use:

```
$ qprt -P lp1 filename
```

To obtain the job number of your print request use the **-#j** option with the **qprt** command at the time of submission.

Print command differences

Alternative commands exist for printing. They are:

AT&T	BSD
\$ lp filename	\$ lpr filename

The following commands are available to list and cancel jobs in the print queues:

AT&T	BSD
\$ lpstat	\$ lpq

Displaying queue information

The **qchk** command by default will only list information about the default queue. To obtain a listing for all the queues defined on your system use the **-A** option or use the **lpstat** command.

The **qcan** command can be used to cancel one file in a queue when used with the **-x** option. It can also be used to cancel all your jobs in a particular queue when used with the **-X** option; that is:

```
$ qcan -X -P lp0
```

Instructor Notes:

Purpose — This demonstration is used to show the students how they can initiate and handle print jobs with AIX printing system. After the demonstration, the students should be able to send a job to the printer and cancel jobs off the queue.

System V Printing System will be covered in the next courses.

Details — If the `qprt` command is used without any options, then the jobs will be submitted to the default queue (which has been set up by the system administrator). If, however, the user wishes to submit a job to another queue defined for their printer then the `-P` option must be used and also the queue name must be known.

Mention the importance of the `-#j` option (introduced in Version 4) which displays the job number of the specified print request. This is very useful as it can be used for cancelling the job off a queue, or even obtaining an update to the status of the request.

If more than one file is specified with the `qprt` command, then the group of files together make up one print job. The files are printed in the order that they are specified on the command line.

Note that before you can print a file, you must have read permission to it.

Using the `qchk` command without any options lists the information about the default queue. Use the `-A` option to get a listing of all the queues on the system (similar to `lpstat`).

To get a listing for a particular job, specify the job number as well, that is,

```
$ qchk -#jobnumber
```

If you specify a job number that does not exist then the system displays the current job number on the queue instead of an error message.

The `qcan` command can be used to cancel off jobs either by specifying the particular job number, or specifying all the jobs in a particular queue. Mention that a user can only cancel off jobs that they have submitted, which is a very desirable feature.

Users can customize their default printer queue by defining the `$PRINTER` environment variable (for example, `export PRINTER=laser`).

Additional Information — The root user can cancel off all jobs on a particular queue using the same `qcan` syntax, that is, `$ qcan -X -P lp0`

Often students will mention that their queue names are truncated when using the `qchk` or `lpstat` commands (that is, only the first seven characters of the queue name are shown). AIX 4.2.1 introduced the `-w` option for the `qchk` command will display up to 20 characters of the queue name. Since AIX V5.1 the `lpstat` command also supports the `-w` option.

In case a student asks....if using a remote printer, two entries for the print queue will actually be shown when using the `qchk` command.

Transition Statement — Let's do a few checkpoint questions before the machine exercises.

Checkpoint

1. What is the effect of the following commands?

```
$ cd /home/team01  
$ cp file1 file2
```

2. What is the effect of the following commands?

```
$ cd /home/team01  
$ mv file1 newfile
```

3. What is the effect of the following commands?

```
$ cd /home/team01  
$ ln newfile myfile
```

4. List commands that can be used to view the contents of a file.

© Copyright IBM Corporation 2008

Figure 5-14. Checkpoint

AU1310.0

Notes:

Instructor Notes:

Purpose — Review the checkpoint questions with the students.

Details —

Checkpoint Solutions

1. What is the effect of the following commands?

```
$ cd /home/team01  
$ cp file1 file2
```

The cp command creates a new file, file 2 from a copy of file1. Each copy will have a different name, as shown, file1 and file2. The two copies are independent of each other. If one file is modified, it does not reflect in the second file.

2. What is the effect of the following commands?

```
$ cd /home/team01  
$ mv file1 newfile
```

These commands will rename file1 to newfile. file1 will no longer exist, but instead be shown as newfile.

3. What is the effect of the following commands?

```
$ cd /home/team01  
$ ln newfile myfile
```

The file called newfile is now known as myfile. An ls -l will show both files. An ls -li will show that both files share the same node number. Note that there is still only one physical file on disk. If a change is made to newfile that change will also be reflected if using myfile.

4. List commands that can be used to view the contents of a file.

cat, pg, more

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — Let's switch over to the lab.

Exercise: Using Files



© Copyright IBM Corporation 2008

Figure 5-15. Exercise: Using Files

AU1310.0

Notes:

After completing the lab, you will be able to:

- Copy, move, link, and remove files
- Display the contents of a file using different commands
- Print a file

Instructor Notes:

Purpose — Transition to the lab.

Details — Provide what students will learn in the lab.

Additional Information —

Transition Statement — Before moving on to another unit, let's review the key points from this unit.

Unit Summary

- The **cp** command can be used to copy files.
- The **mv** command can be used to move and rename files.
- The **ln** command can be used to create additional names for a file.
- Display the contents of a file using **cat**, **pg**, or **more**.
- Use the **rm** command to delete files.
- Use the **qprt** command to print files.
- The **wc** command could be used to count words or lines from files or command output.

© Copyright IBM Corporation 2008

Figure 5-16. Unit Summary

AU1310.0

Notes:

Instructor Notes:

Purpose — Describe the key points from this unit's material.

Details —

Additional Information —

Transition Statement — Let's move on to another unit.

Unit 6. File Permissions

Estimated Time

01:00

What This Unit Is About

This unit introduces the students to the concept of protecting files from unauthorized access by controlling a file's permissions.

What You Should Be Able to Do

After completing this unit, students should be able to:

- List the basic file permissions
- Change the basic file permissions using both the octal and symbolic formats

How You Will Check Your Progress

Accountability:

- Student Activity
- Checkpoint questions
- Exercise

Unit Objectives

After completing this unit, you should be able to:

- List the basic file permissions
- Change the basic file permissions using both the octal and symbolic formats

© Copyright IBM Corporation 2008

Figure 6-1. Unit Objectives

AU1310.0

Notes:

Instructor Notes:

Purpose — To introduce the unit objectives.

Details —

Additional Information —

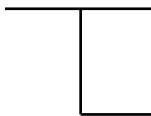
Transition Statement — How can you see what permissions are assigned to a file?

Long Listing of Files

The **ls** command with the **-l** option can be used to obtain more information about the files in a directory.

```
$ ls -l
```

drwxrwxr-x	2	team01	staff	1024	Aug 12	10:16	c
drwxrwxr-x	2	team01	staff	512	Feb 18	09:55	doc
-rwxrwxr-x	1	team01	staff	320	Feb 22	07:30	suba
-rwxrwxr-x	2	team01	staff	144	Feb 22	16:30	test1



→ Permission Bits

© Copyright IBM Corporation 2008

Figure 6-2. Long Listing of Files

AU1310.0

Notes:

Introduction

An i-node describes a file or directory entry as it appears on a disk. Each file has one i-node assigned to it. The information that is displayed from the **ls -l** command (as shown on the above visual) is read from the i-nodes associated with the files listed.

ls command output

The fields from the `ls -l` command are as follows:

(1)	(2)	(3)	(4)	(5)	(6)	(7)
drwxrwxr-x	2	team01	staff	1024	Aug 12 10:16	c
drwxrwxr-x	2	team01	staff	512	Feb 18 09:55	doc
-rwxrwxr-x	1	team01	staff	320	Feb 22 07:30	suba
-rwxrwxr-x	2	team01	staff	144	Feb 22 16:30	test1

- Field 1 shows the file/directory and permission bits
- Field 2 is the link count
- Field 3 shows the user name of the person who owns entry
- Field 4 shows the name of the group for which group protection privileges are in effect
- Field 5 shows the character count of the entry
- Field 6 shows the date and time the file was last modified
- Field 7 shows the name of the file/directory

The `-d` option used with the `-l` option of the `ls` command is another very useful option. The `-d` option will display only the information about the directory specified. Directories are treated like ordinary files.

Instructor Notes:

Purpose — The visual shows the output obtained from the `ls -l` command.

Details — The `ls -l` command has been discussed previously. In this unit, we are going to focus on the information shown in the first column, the permission bits.

You may also want to review the information shown in column two, the link count.

Field 2 Shows the link count. Remind the students that they should know how to create a linked file with the `ln` command. If a linked file is created, then the link count is incremented by 1. Ordinary files by default have a link count of 1, whereas directories have a count of 2.

Additional Information —

Field 2 When explaining link counts, you could also mention that when the `rm` command is issued, the link count is decremented by 1, and only when the link count is equal to 0, will a file be removed.

Transition Statement — Now, let's turn our attention to the permission bits.

File Protection/Permissions

- Every file and directory on the system has file permissions associated with it.
- Three permission categories: **owner**, **group**, and **other**
- Three bits can be set for each category: **read, write, execute (rwx)**

- For an **ordinary** file:

`r` => Can look at the contents of a file
`w` => Can change or delete the contents of a file
`x` => Can execute the file (`r` is also needed if a script)

- For a **directory**:

`r` => Can list the files within a directory (`ls`)
`w` => Can modify/remove files in the directory
`x` => Can change into the directory and access the files within (`cd`)

© Copyright IBM Corporation 2008

Figure 6-3. File Protection/Permissions

AU1310.0

Notes:

File permissions

The visual describes the three permission categories, and the associated permission bits that can be set for each.

For files, the permission bits are straightforward. For shell scripts, the `r` bit is required with the `x` to make the file executable. Compiled (binary) programs only require the `x`.

For directories, the `x` bit is required to access any of the files or subdirectories within it. This implies that the `x` bit is required on all directories above it as well.

The `w` bit on a directory overrides any permissions on the files within that directory. Thus, if a user has write access to a directory, they can remove ANY files or directories within it, regardless of the individual file's permissions.

Instructor Notes:

Purpose — To show the base permission **r**, **w**, and **x** for ordinary files and directories.

Details — Refer to the output from the **ls -l** command and highlight how the nine bits are broken into three groups of three. Explain what each group refers to. Next, go through each permission explaining what is gained by having that bit set on first for a file and then for a directory.

Note that withholding execute permission on a directory will allow an **ls** listing, but not an “**ls -l**” listing.

Discussion Items — You could also use **ls -al**. Why would you want to?

Answer: This will give you the long listing for the hidden files too.

Additional Information — You may wish to mention that merely withholding execute permission on a script or command does little to protect anything. Someone can still write and execute their own code to do the same function. If a user has read access to the file, they can make a copy of it (in their home directory) and add the execute permission themselves.

There is a fourth grouping of permissions which controls the “set” bits: SUID, SGID, and SVTX (or sticky). These are outside the scope of this course, but you should understand their use and management and be prepared to handle student questions about them (offline, during break, and so on).

Transition Statement — Now that we understand the different permissions that can be set, let's see how we can change them.

Changing Permissions (Symbolic Notation)

chmod mode filename

u = owner of the file	+ : add permissions
g = owner's group	- : remove permissions
o = other users on the system	= : clears permissions and sets to mode specified
a = all	

```
$ ls -l newfile
-rw-r--r-- 1 team01 staff 58 Apr 21 16:06 newfile

$ chmod go+rw newfile
$ ls -l newfile
-rw-rw-rw- 1 team01 staff 58 Apr 21 16:06 newfile

$ chmod a+tx newfile
$ ls -l newfile
-rwxrwxrwx 1 team01 staff 58 Apr 21 16:06 newfile

$ chmod o-rwx newfile
$ ls -l newfile
-rwxrwx--- 1 team01 staff 58 Apr 21 16:06 newfile
```

© Copyright IBM Corporation 2008

Figure 6-4. Changing Permissions (Symbolic Notation)

AU1310.0

Notes:

Symbolic notation

With symbolic notation, you are specifying changes relative to the existing permissions on a file or directory by adding or deleting permissions. You can check what the permissions are currently set to by using the **ls -l** command.

You can specify multiple symbolic modes separated with commas. Do not separate items in this list with spaces. Operations are performed in the order they appear from left to right.

When you use the Symbolic mode to specify permission modes, the first set of parameters selects the permission field, as follows:

- u** File owner
- g** Group
- o** All others

- a User, group, and all others. This has the same effect as specifying the *ugo* options. The a option is the default permission field. If the permission field is omitted, the default is the a option.

The second set of flags selects whether permissions are to be taken away, added, or set exactly as specified:

- Removes specified permissions
- + Adds specified permissions
- = Clears the selected permission field and sets it to the mode specified. If you do not specify a permission mode following =, the **chmod** command removes all permissions from the selected field.

The third set of parameters of the **chmod** command selects the permissions as follows:

- r Read permission
- w Write permission
- x Execute permission for files; search permission for directories.

Instructor Notes:

Purpose — To illustrate how the symbolic method can be used to change base permissions.

Details — Explain each symbol as detailed in the student notes.

The `chmod` command accepts comma delimited lists of operations as the first parameter. See the following example, of setting the permissions of a file to `-rwxrw-r--`:

```
chmod u=rwx,g=rw,o=r testfile
```

No spaces are allowed between the commas.

Setting permissions using the symbolic method will leave any ACLs on the file intact.

Additional Information —

Transition Statement — Let's now consider using the octal notation.

Changing Permissions (Octal Notation)

- File and directory permissions can be specified in the symbolic syntax or as an **octal number**:

	User	Group	Others
Symbolic	rwx	rw-	r--
Binary	111	110	100
	4+2+1	4+2+0	4+0+0
Octal	7	6	4

- To change permissions so the **owner** and **group** have **read** and **write** permissions and **others** **read only**:

```
$ ls -l newfile
-rw-r--r-- 1 team01 staff 58 Apr 21 16:06 newfile

$ chmod 664 newfile
$ ls -l newfile
-rw-rw-r-- 1 team01 staff 65 Apr 22 17:06 newfile
```

© Copyright IBM Corporation 2008

Figure 6-5. Changing Permissions (Octal Notation)

AU1310.0

Notes:

Permission notation

Each permission in the group of nine is represented by a one and a lack of permission is represented by a zero. So rw-r--r-- translates to 110100100 in binary, or 644 in octal notation.

When using octal notation, all permission bits must be set; you cannot change permissions on one category (that is, just the owner, or just the group).

Translating permissions

The chart below may help in translating binary to octal for those who are unfamiliar with binary notation:

user			group			others		
r	w	x	r	w	x	r	w	x
400			40			4		
200				20			2	
		100			10			1

In order to translate the mode you require to a number, add the numbers corresponding to the permissions you need. So, if you need the file to be readable and writable by the owner and group, and readable by all the other users of the system, simply perform the addition:

400
 200
 40
 20
 4

 664

The `chmod` command would be:

`$ chmod 664 newfile`

With the octal format, you specify a file's final permissions.

Warning messages

Sometimes the file permission will generate a safety prompt, rather than totally preventing you from completing the operation. For example, if you are the owner of a file, you have no permissions on that file (for example 000), and you try to remove it, the system will ask you if you want to override the protection setting on the file that you wish to remove. You may respond yes at this point, and the system will remove your file. The same will happen if you are a member of the group.

Instructor Notes:

Purpose — To illustrate how the octal format can be used to change permissions.

Details — The octal format at first glance may seem easier, however there are some additional permissions (which should not be discussed in this course) which cannot be unset using the octal method. For example, you cannot remove the `SETGID` attribute on a directory using the octal notation. To remove the `SETGID` attribute on a directory, you must explicitly remove the permission using the symbolic mode form.

Additional Information —

Transition Statement — Let's see what permissions are obtained when a new file or directory is created.

Default File Permissions

The default protections for newly created files and directories are:

File	<code>-rw-r--r--</code>	644
------	-------------------------	------------

Directory	<code>drwxr-xr-x</code>	755
-----------	-------------------------	------------

These default settings may be changed by changing the [umask](#) value.

© Copyright IBM Corporation 2008

Figure 6-6. Default File Permissions

AU1310.0

Notes:

System defaults

The real default permissions for a file and directory are 666 and 777 respectively. The [umask](#) value is then subtracted from these values. The default [umask](#) is 022, which leaves you with values of 644 for a file and 755 for a directory.

Instructor Notes:

Purpose — To define the default permissions for files and directories.

Details —

Additional Information —

Transition Statement — Let's take a closer look at the `umask` value.

umask

- The **umask** specifies what permission bits will be set on a **new file** or **directory** when created. It is an **octal number** that is used to determine what permission bits a file or directory is created with:

New Directory: 777 - 022: **755** => **rwxr-xr-x**

New File: 666 - 022: **644** => **rw-r--r--**

- The default value of **022** is set in **/etc/security/user**. It can be changed for all users or for a specific user.

© Copyright IBM Corporation 2008

Figure 6-7. umask

AU1310.0

Notes:

Understanding permissions

umask is a command which sets the **umask** value by accepting an octal permission string as an argument. If an argument is not provided then **umask** will display the existing **umask** value.

The **chmod** command works by *applying* a permissions mask onto a file. **umask**, on the other hand, works by *taking away* these permissions.

The default setting of the **umask** is 022. For tighter security, you should make the **umask** 027 or even 077.

A **umask** of 022 specifies that the permissions on a new file will be 644 or on a new directory will be 755. A **umask** of 000 would give 666 permissions on a file (read/write access to all) or 777 on a directory (read/write/execute access to all).

Execute permissions are never set when ordinary files are created.

Permission octal format

Remember, the permissions, in octal, are:

0	0	0	= nothing
1	1	1	= eXecute
2	2	2	= Write
4	4	4	= Read

_____ user _____ group _____ others

Using `chmod`, permissions are granted by summing the octal values for each category (user, group or others), for example 644 means (2+4)(4)(4) or (w+r)(r)(r).

Instructor Notes:

Purpose — The `umask` is responsible for determining what permissions a file or a directory will receive upon creation.

Details — Remember the permissions in octal, are:

0	0	0	= nothing
1	1	1	= eXecute
2	2	2	= Write
4	4	4	= Read
user	group	others	

Using `chmod`, permissions are granted by summing the octal values for each category (user, group or others), for example 644 means (2+4)(4)(4) or (w+r)(r)(r). `umask` works in the opposite manner to `chmod` by specifying those permissions to remove by default. The default `umask` is 022, or (nothing)(write)(write), which produces files like:

`-rw-r--r--`

and directories like:

`drwxr-xr-x`

If this were changed to, for instance, 027, then files would result like:

`-rw-r-----`

and directories like:

`drwxr-x---`

By manipulating the `umask`, a user can ensure that, by default, all their files and directories have the correct access controls.

Where can we use the `umask` to change permission for all users or for some users?

- For *all users*, it should be done in `/etc/security/user` in the default stanza. It also can be done in the `/etc/profile` by entering command line into it, and this will override the previous umask set in `/etc/security/user`. So it can be done in `/etc/security/user`.
- For an *individual user*, it should be done in `/etc/security/users` in his stanza. It also can be done in the `$HOME/.profile` by entering command line into it, and this will override the previous umask set in `/etc/security/user`.
- For *temporarily* changes, an *individual user* can enter the `umask` command on the command line.

Additional Information — `chmod` and `umask` are described in the manual pages as well as a more detailed description in the online documentation.

Note: On a file, the execute permission is never set by default.

Transition Statement — Let's do a short activity.

Activity: Personal Directories



© Copyright IBM Corporation 2008

Figure 6-8. Activity: Personal Directories

AU1310.0

Notes:

Activity

In this activity you will review the **umask** and **chmod** commands.

___ 1. Log in to the system.

___ 2. Execute the **umask** command and write down the **umask** you are using:

___ 3. According to your **umask**, what default file permission do you expect for a new directory or a new file?

New directory: _____

New file: _____

-
- ___ 4. Create a new directory **testdir1** and check the file permissions.
 - ___ 5. Create a new file **testfile1** and check the file permissions.
 - ___ 6. Execute the command **umask 027** to change your default umask.
 - ___ 7. Create a new directory **personal** and check the file permissions. What difference do you see?
-
- ___ 8. In a private directory where personal files are stored, you should prevent others from accessing this directory. Execute the **chmod** command and protect your **personal** directory.

Write down the command you executed:

- ___ 9. Execute **ls -ld personal** and check that the rights are correct.

Please reset the **umask** to the value found in step 2 or log out and log in again.

Optional activity:

- ___ 10. Verify with the **tty** command on which terminal you are working. Display the permissions of that terminal with the command **ls -l \$(tty)**. Now use the command **mesg** with option **y** or **n** to allow or deny messages via **write** or **wall** commands to this terminal. Display the permissions again. What does the **mesg** command do?

Instructor Notes:

Purpose — To test the students' understanding of the material presented so far.

Details — Let students review the `umask` and `chmod` commands.

Optional activity:

___10. Answer: The `mesg` command works like a `chmod` command for the current terminal.

Additional Information —

Transition Statement — Let's summarize the permissions necessary to execute various AIX commands.

Function/Permissions Required

Command	Source Directory	Source File	Target Directory
<code>cd</code>	x	N/A	N/A
<code>ls</code>	r	N/A	N/A
<code>ls -l</code>	r, x	N/A	N/A
<code>mkdir</code>	x w (parent)	N/A	N/A
<code>rmdir</code>	x w (parent)	N/A	N/A
<code>cat, pg, more</code>	x	r	N/A
<code>mv</code>	X, W	NONE	X, W
<code>cp</code>	x	r	X, W
<code>touch</code>	x, w *	NONE	N/A
<code>rm</code>	x, w	NONE	N/A

© Copyright IBM Corporation 2008

Figure 6-9. Function/Permissions Required

AU1310.0

Notes:

File permissions

This table can be used as a reference to ensure that the correct permissions are set on files and directories to accomplish the desired activity.

* `w` permission is also needed in the source directory when using the `touch` command to create a zero-length file. `w` permission is *not* necessary if using the `touch` command on an existing file for the purpose of updating the modification date.

Instructor Notes:

Purpose — To establish what permissions are required to carry out the commands which we have discussed so far.

Details — This table should be used as reference material by the students to ensure that the correct permissions are set on files and directories to accomplish the desired activity.

Point out that to remove a file you do not have to have write permissions on the file, only on the directory. This is often overlooked by the student who only considers permissions set at the local file level.

Additional Information —

Transition Statement — Before trying this out on the systems, a few checkpoint questions.

Checkpoint (1 of 3)

The following questions are for a file called **reporta** which has the following set of permissions: **rwxr-x r-x**

1. What is the mode in octal?
2. Change mode to **rwxr--r--** using symbolic format.
3. Repeat the above operation using octal format.

Question four is based on the following listing. Assume that the directory **jobs** contains the file **joblog**.

```
$ ls -lR
total 8
drwxr-xr-x  2  judy  finance  512  June  5  11:08  jobs
./jobs:
total 8
-rw-rw-r--  1  judy  finance   100  June  6  12:16  joblog
```

4. Can Fred, who is a member of the finance group, modify the file **joblog**?

© Copyright IBM Corporation 2008

Figure 6-10. Checkpoint (1 of 3)

AU1310.0

Notes:

Instructor Notes:

Purpose — To test the students' knowledge of the presented material.

Details —

Checkpoint Solutions (1 of 3)

The following questions are for a file called **reporta** which has the following set of permissions: **rwxr-x r-x**

1. What is the mode in octal? **755**
2. Change mode to **rwxr--r--** using symbolic format. **chmod go-x reporta**
3. Repeat the above operation using octal format. **chmod 744 reporta**

Question four is based on the following listing. Assume that the directory **jobs** contains the file **joblog**.

```
$ ls -lR
total 8
drwxr-xr-x  2 judy  finance  512  June  5  11:08  jobs
./jobs:
total 8
-rw-rw-r--  1 judy  finance   100  June  6  12:16  joblog
```

4. Can Fred, who is a member of the finance group, modify the file **joblog**? **Yes, he can, as the file has write permission on the file and has execute permission on the directory.**

© Copyright IBM Corporation 2008

Additional Information — There are many ways to change permission with symbolic format and the one shown is only one way. Other ways could be:

```
chmod u=rwx,go=r reporta
chmod ugo=r,u+wx reporta
```

Transition Statement — A few more questions...

Checkpoint (2 of 3)

This question is based on the following listing. Assume that the directory **jobs** contains the directory **work**, which in turn contains the file **joblog**.

```
$ ls -lR
total 8
drwxrwxr-x  3  judy   finance  512  June  5  11:08  jobs

./jobs:
total 8
drwxrw-r-x  2  judy   finance  512  June  5  11:10  work

./jobs/work:
total 8
-rw-rw-r--  1  judy   finance  100  June  6  12:16  joblog
```

5. Can Fred, who is a member of the finance group, modify the file **joblog**?

© Copyright IBM Corporation 2008

Figure 6-11. Checkpoint (2 of 3)

AU1310.0

Notes:

Instructor Notes:

Purpose — To test the students' knowledge of the presented material.

Details —

Checkpoint Solutions (2 of 3)

This question is based on the following listing. Assume that the directory **jobs** contains the directory **work**, which in turn contains the file **joblog**.

```
$ ls -lR
total 8
drwxrwxr-x  3  judy  finance  512  June  5  11:08  jobs

./jobs:
total 8
drwxrwxr-x  2  judy  finance  512  June  5  11:10  work

./jobs/work:
total 8
-rw-rw-r--  1  judy  finance   100  June  6  12:16  joblog
```

5. Can Fred, who is a member of the finance group, modify the file **joblog**? **No, because he does not have execute permission on the intermediate directory, work.**

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — A few more questions...

Checkpoint (3 of 3)

This question is based on the following listing. Assume that the directory **jobs** contains the directory **work**, which in turn contains the file **joblog**.

```
$ ls -lR
total 8
drwxr-xr-x  3  judy   finance  512  June  5  11:08  jobs

./jobs:
total 8
drwxrwxrwx  2  judy   finance  512  June  5  11:10  work

./jobs/work:
total 8
-rw-rw-r--  1  judy   finance   100  June  6  12:16  joblog
```

6. Can Fred, who is a member of the finance group, copy the file **joblog** to his home directory?

© Copyright IBM Corporation 2008

Figure 6-12. Checkpoint (3 of 3)

AU1310.0

Notes:

Instructor Notes:

Purpose — To test the students' knowledge of the presented material.

Details —

Checkpoint Solutions (3 of 3)

This question is based on the following listing. Assume that the directory **jobs** contains the directory **work**, which in turn contains the file **joblog**.

```
$ ls -lR
total 8
drwxr-xr-x  3  judy  finance  512  June  5  11:08  jobs

./jobs:
total 8
drwxrwxrwx  2  judy  finance  512  June  5  11:10  work

./jobs/work:
total 8
-rw-rw-r--  1  judy  finance   100  June  6  12:16  joblog
```

6. Can Fred, who is a member of the finance group, copy the file **joblog** to his home directory? **Yes**.

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — Now let's test your understanding with a machine exercise.

Exercise: File Permissions



© Copyright IBM Corporation 2008

Figure 6-13. Exercise: File Permissions

AU1310.0

Notes:

After completing the exercise, you will be able to:

- Manipulate permissions on ordinary files and directories
- Interpret file and directory permission bits

Instructor Notes:

Purpose — Transition to the lab.

Details — Describe what the students will learn in the lab exercise.

Additional Information —

Transition Statement — And finally, I'd like to summarize this unit's key points.

Unit Summary

- Basic file permissions can be listed using the `ls -l` command.
- `chmod` grants or removes read, write, and execute permissions for three classes of users: `user`, `group` and `others`.
- The permissions used with the `chmod` command can be defined in symbolic or octal format.
- The `umask` specifies the permissions for new files and directories.

© Copyright IBM Corporation 2008

Figure 6-14. Unit Summary

AU1310.0

Notes:

Instructor Notes:

Purpose — Summarize this unit's key points.

Details —

Additional Information —

Transition Statement — Let's move on to the next unit.

Unit 7. The vi Editor

Estimated Time

00:45

What This Unit Is About

This unit is an introduction to the `vi` editor. It describes how to begin an edit session, add text, remove text, and save text within a file.

What You Should Be Able to Do

After completing this unit, students should be able to:

- Create and edit files
- Manipulate text within a file
- Set up defaults for the `vi` editor
- Execute command line editing
- Define the uses for the other forms of `vi`

How You Will Check Your Progress

Accountability:

- Student activity
- Checkpoint questions
- Exercise

Unit Objectives

After completing this unit, you should be able to:

- Create and edit files
- Manipulate text within a file
- Set up defaults for the vi editor
- Execute command line editing
- Define the uses for the other forms of vi

© Copyright IBM Corporation 2008

Figure 7-1. Unit Objectives

AU1310.0

Notes:

Instructor Notes:

Purpose — To introduce the unit objectives.

Details —

Additional Information —

Transition Statement — Let's start an overview of the `vi` editor.

Introduction to the vi Editor

- **Full-screen** editor
- **Two modes** of operation: **command** and **text**
- Utilizes **one letter commands**
- Does not format text
- Flexible search and replace facility with **pattern matching**
- Allows for user-defined editing functions using **macros**

© Copyright IBM Corporation 2008

Figure 7-2. Introduction to the vi Editor

AU1310.0

Notes:

The vi editor

It is important to know **vi** for the following reasons:

- It is the only editor available in maintenance mode on IBM System p
- Standard editor across all UNIX systems
- Command line editing feature
- Used as default editor for some programs

Modes of operation

The **vi** editor has two modes of operation. *Command mode* allows the user to perform operations on the text, such as cut, paste, cursor movement, and replacement. These operations are all invoked by one character “commands.” *Text mode* (also known as *insert mode*) allows the user to enter text; all characters typed are inserted into the file.

There is no way to tell which mode the editor is in, other than pressing a key on the keyboard. If the key you press is inserted into the file, the current mode is text mode. If the key performs an action, the current mode is command mode.

Switching from command mode to text mode is done with one of the insert text commands, which will be covered later. Returning to command mode is done by pressing the <Esc> key.

Advanced features

The **vi** editor has many advanced features, such as flexible search and replace (using standard regular expressions), undo, and user-defined editing functions (called *macros*).

It is important to remember that the **vi** editor edits ASCII text files directly; it does not format the text in any way, as a word processor would do.

Introduction to vi functions

This unit covers only a subset of the **vi** functions. It is a very powerful editor. Refer to the online documentation for additional functions. Refer to the *Command Summary* in the appendices for a reference guide on using **vi**.

Instructor Notes:

Purpose — Describe the `vi` editor, how it works and its modes of operation.

Additional Information — Make very clear the modes of operation in `vi`. By default, there is no way of telling what mode is currently active unless a command is used or the escape key is pressed.

At this point, you can point out that there are different *modes* in `vi` and that these can be set as a new default. This will be discussed later in the unit.

Transition Statement — Let us now see how to invoke the editor.

Starting vi

```
$ vi vifile
```

- If the file "**vifile**" does not exist, it will be created
 - Otherwise, **vi** will open the existing file

© Copyright IBM Corporation 2008

Figure 7-3. Starting vi

AU1310.0

Notes:

Editor startup

When the editor starts up, it needs to use work space for the new or existing file you are going to edit. It does this by using an editing buffer. When a session is initiated, one of two things happens:

- If the file to be edited exists, a copy of the file is put into a buffer in **/var/tmp** by default.
 - If the file does not exist, an empty buffer is opened for this session.

The tilde characters represent empty lines in the editor.

The editor starts in command mode.

Instructor Notes:

Purpose — Show students how to invoke the `vi` editor.

Discussion Items — Ask a question here: What will happen if you are in the root directory and begin an edit session with a filename that does not exist in that directory, with the intention of creating it?

(Answer: Nothing, no warnings about permissions!)

Transition Statement — Having started `vi`, let's see how we can enter a few lines in a file.

Adding Text

\$ vi vifile

keystroke

i

This file is being created using the vi editor.
 To learn more about the vi editor, look in the "Commands Reference" manual under vi.

~
 ~
 ~
 ~
 ~
 ~
 ~
 ~
 ~
 ~
 ~

© Copyright IBM Corporation 2008

Figure 7-4. Adding Text

AU1310.0

Notes:

Adding text to a file

To type text into a file, the following commands can be used:

a	append text after the cursor
A	append text to the end of the line
i	insert text at the cursor
I	insert text at the start of the line
o	open a new line under the current line
O	open a new line above the current line

Each of these commands places the session into text mode. Any characters entered will be placed into the file.

To exit from text mode, press the **<Esc>** key.

Instructor Notes:

Purpose — This visual demonstrates how text can be added.

Details — Explain the differences between the **i** and the **a** commands. **i** will insert text before the current position of the cursor, whereas **a** will append text after the cursor.

Transition Statement — Having added text, let's see how we can leave **vi**.

Exiting the Editor

```
$ vi vifile
```

keystroke:

<Esc>

This file is being created using the vi editor.
To learn more about the vi editor,
look in the "Commands Reference" manual
under vi.

卷之三

:wq

- To quit **without** saving: :q!
 - To **save** and **exit**: :x or :wq or <shift-zz>

© Copyright IBM Corporation 2008

Figure 7-5. Exiting the Editor

AU1310.0

Notes:

Exiting the editor

To get into command mode, or to ensure that you are in command mode, press `<Esc>` before carrying out any commands.

These commands will exit the editor. Each will exit differently.

:q quits without saving. This option will only work if you have not made any changes. If you have made changes, then to force an exit out of the editor, use ! with the q command.

:w writes changes

:x saves and exits

:wq writes changes and quits

<Shift+zz> writes changes and quits

The “`:`” subcommand will place the cursor at the bottom of the screen.

Instructor Notes:

Purpose — To illustrate the different methods of exiting from the **vi** editor.

Details — There may be times when **vi** will flash a message on screen stating that the user must force an exit.

This means the above commands (**:q!**, **:wq!**, and so forth) must be used with the **!** character:

:q! to force the quit

:wq! to force the write and quit

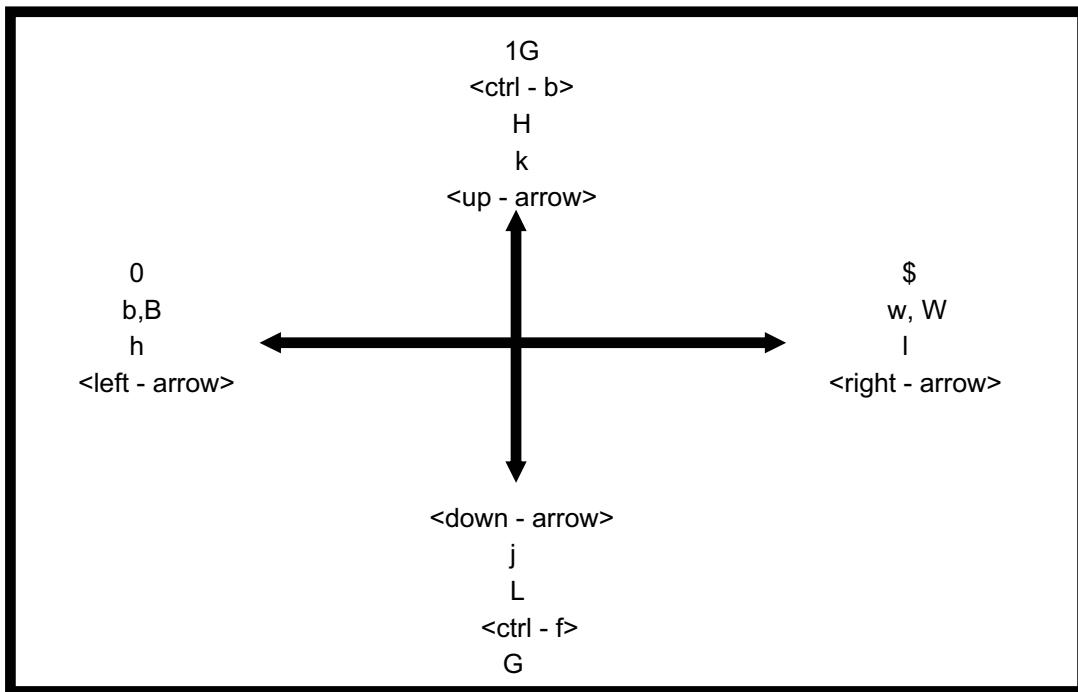
Discussion Items — A question concerning file and directory permissions can be asked here:

Like on the previous visual, what will happen if user **teamxx** tries to create and save a file in the root directory?

(Answer: Cannot create it!)

Transition Statement — We will now see how to move around the screen or file.

Cursor Movement



© Copyright IBM Corporation 2008

Figure 7-6. Cursor Movement

AU1310.0

Notes:

Moving within a file

Note that for all the uppercase specified commands, the **<shift>** key must be used.
The following commands describe different techniques for moving around within a file.

Moving within a line

To move within a line make sure you are in command mode and:

<left-arrow> or h	one character left
<right-arrow> or l	one character right
0	move to beginning of line
\$	move to end of line

Moving to a word

To move to a word:

w	next small word
W	next big word
b	back to previous small word
B	back to previous big word
e	end of next small word
E	end of next big word

Moving within the screen

To move within the screen:

<up-arrow> or k	one line up
<down-arrow> or j	one line down
H	top line on screen
M	middle line on screen
L	last line on screen

Scrolling the screen

To scroll the screen:

<ctrl-f>	scroll forward
<ctrl-b>	scroll backward

Moving within the file

To move within the file:

1G	go to the first line
45G	go to line number 45
G	go to the last line

Instructor Notes:

Purpose — Show the various methods of cursor movement in **vi**.

Details — The diagram shows ways of moving around. It also shows these in increments of motion.

For example:

<down-arrow> or j	go down one line
L	go to last line on the screen
<ctrl-f>	go forward to the next screen
G	go to the last line in the file

Depending on the class pace and timing, you might be selective in choosing which commands to cover. While you should be prepared to explain any of the commands (for example the difference between moving over a “big word” vs. a “small word”), most students will be overwhelmed by covering them all. At minimum they should master moving in each direction one character at a time.

Additional Information — There are other commands to move around. These are documented in the manual pages and the Web-based documentation.

Note that the ability to use the keyboard cursor control keys of <up-arrow>, <down-arrow>, <left arrow>, and <right arrow> will depend on the terminal or terminal emulation being used.

Transition Statement — Let's see how we can delete text from a file.

Deleting Text

To delete a single character :	x
To delete to the end of the current word :	dw
To delete to the end of the line :	d\$
To delete to the start of the line :	d0
To delete the whole line :	dd
To delete a range of lines :	:20,40d
Undo the last change:	u

© Copyright IBM Corporation 2008

Figure 7-7. Deleting Text

AU1310.0

Notes:

Introduction

To execute any of the illustrated commands, you must be in command mode.

There are several different ways to perform the delete functions. See the AIX documentation for other **vi** delete functions.

Instructor Notes:

Purpose — Show the student the many commands available for deleting text.

Details — There are many more commands not mentioned in the visual.

It is counterproductive to give students long command lists. The information is in the online documentation if they need to know more.

One point about this visual is to demonstrate, once again, that **vi** is a very powerful editor indeed.

In case of accidental deletion of text, changes can be undeleted or undone.

- ✉ undo the last change
- ✉ restore the line if the cursor has not left the line

Additional Information — The **vi** editor only has one buffer in which to hold the previous contents that have been modified. When you do a new text change, it overwrite the existing buffer with the new previous contents related to that change. As a result the undo command can only undo the immediate previous operation.

Transition Statement — Having seen how to add and delete text, let us now look at searching through files for text.

Search for a Pattern

\$ **vi vifile**

keystroke:

<Esc>

n

```
This file is being created using the
vi editor.
To learn more about the vi editor,
look in the "Commands Reference" manual
under vi.

~
~
~
~
~
~
~
~
~
~
~
/
/the
```

- To search forward, use **/text**
- To search backward, use **?text**

© Copyright IBM Corporation 2008

Figure 7-8. Search for a Pattern

AU1310.0

Notes:

Searching for patterns

When in command mode, pressing / automatically puts you at the bottom of the file, ready to type in a search pattern.

/the searches forward for the first occurrence of the word **the**.

?the searches backward for the first occurrence of the word **the**.

Pressing the **n** key will continue the search in the same direction (forward if / was used, backward if ? was used).

The **N** key will continue the search in the opposite direction.

Instructor Notes:

Purpose — This visual demonstrates how to search for text in **vi**.

Details — The search can be repeated forward by using the **n** command. If the uppercase command **N** is used, then the direction of the search is reversed.

Additional Information — Remind the students that, by default, the search done here is case-sensitive.

Transition Statement — Let's do a short activity.

Activity: vi Commands

→ Assign the following vi commands:

a, i, u, x, dd, G, 1G, ESC, :q!, :wq

Quit without saving:	
Delete the whole line:	
Exit from text mode:	
Add text after cursor:	
Undo the last change:	
Go to the last line:	
Delete a single character:	
Insert text at the cursor:	
Write changes and quit:	
Go to the first line:	

© Copyright IBM Corporation 2008

Figure 7-9. Activity: vi Commands

AU1310.0

Notes:

Complete the table in the visual.

Instructor Notes:

Purpose — To see how well the students remember the commands presented so far in this unit.

Details — Give the students some time to review the `vi` commands that have been taught. Then cover the solutions.

Additional Information —

Transition Statement — Let's explain how to change text.

Changing Text

\$ **vi vifile**

keystroke:

<Esc>

This file is being created using the vi editor.

To learn more about the one and only vi editor,
look in the one and only "Commands Reference"
manual under vi.

~
~
~
~
~
~
~

:g/ **the** /s// **the one and only** /g

- Text can be replaced by overtyping: **Rnewtext**
- A single character can be replaced: **rX (X=new character)**
- Words can be changed: **c2w**
- Or every occurrence of a word can be substituted for another word or words

© Copyright IBM Corporation 2008

Figure 7-10. Changing Text

AU1310.0

Notes:

Changing text

The command :g/ **the** /s// **the one and only** /g finds every occurrence of the string **the** and replaces it with **the one and only**. Notice that it would not replace the **the** on the first line of text. Remember about spaces: there was no space between the **the** and the new line character.

The first **g** tells the system to search for the first occurrence of the string on every line in the file.

The **s** stands for substitute. The next two slashes direct the editor to use the search string used in the preceding command, in this example the string **the** and replace it with the string **the one and only**.

The last **g** stands for *global* and directs the change to be made at every occurrence across each line being searched.

The **r** command will place you in text mode and allow you to overtype the existing text beginning at the current cursor position. The **x** command will place you in text mode and allow you to overtype only the letter at the current cursor position; after replacing that one letter you are placed immediately back into command mode.

The **c** command can be used to specify the scope of what you want to replace, so that if you want to replace two words spanning a total of 10 characters, you can replace them with a very long string, such as 20 characters, and not overwrite the words which follow those two words.

Instructor Notes:

Purpose — This visual shows a few ways in which text can be changed.

Details — Yet again, there are MANY more ways of doing this. The intention is to just display some examples. Make sure the reason why the first occurrence of the pattern `the` (on the first line) is not replaced, is explained fully.

Discussion Items — A question may be asked here concerning searching and replacing text:

In the example above, where the letters `the` are replaced by the string `the one and only`, what will happen if the document contains words like: `therefore` and `further`, and so forth?

If the command was written:

```
:g/the/s//the one and only/g
```

then the result would be:

the one and onlyrefore	in the therefore case
furthe one and onlyr	in the further case

So, how do you work around this?

Answer:

Put some spaces in to delimit the word `the`!

```
:g/ the /s// the one and only /g
```

Also, adding a `c` at the end of this command will cause a confirmation to be displayed. For example,

```
:g/ the /s// the one and only /cg
```

Additional Information — For more information on global text operations, see the documentation on the `sed` or `ed` facilities.

Transition Statement — Changing text works fine when the original is no longer wanted. We now look at copying and moving text.

Cut, Copy, and Paste

Commands	
<pre>This is the first line of text - This is the second line of text This is the third line of text</pre>	dd
<pre>This is the first line of text - This is the third line of text</pre>	
<pre>This is the first line of text This is the third line of text - This is the second line of text</pre>	p
<ul style="list-style-type: none"> • To cut text, use the d command. • To copy text, use the y command. • To paste the cut or copied text on the line below the cursor position, use the p command. 	

© Copyright IBM Corporation 2008

Figure 7-11. Cut, Copy, and Paste

AU1310.0

Notes:

How to copy, cut, and paste

To copy into a temporary buffer:

yy	places the current line into a buffer
-----------	---------------------------------------

To cut (or move) text:

dd	delete the current line (and store it in the undo buffer)
10dd	delete the next 10 lines (and place them in the undo buffer)

To put text back:

p	puts text back after the cursor or on the next line
P	puts text back before the cursor or on the previous line

The **u** command will UNDO your last command if you make an error. So, if you delete something in error, immediately type the **u** command to retrieve it.

Cutting and copying multiple lines

Multiple lines can cut or copied by specifying a number either in front of the **dd** or **yy** command, or between the letters of the command.

10dd (or d10d) Cut 10 lines from the cursor position down into the buffer.

15yy (or y15y) Copy 15 lines from the cursor position down into the buffer.

Instructor Notes:

Purpose — This visual shows how to copy, cut, and paste.

Details —

Additional Information — There are yank buffers 0-9 and a-z. Deleted or yanked data will by default be placed into buffer 0. This can be overridden by:

"**a20yy** to yank the next 20 lines into yank buffer a

or

"**2dd** to delete the current line and place its contents in buffer number 2.

To put the yanked data back, you can use:

p to paste the information above the current line

P to paste the information below the current line

The alternative is to UNDO the deletion, use the **u** or **undo** command.

(In the examples above, " is a double quote and not two single quotes.)

The **u** command will undo all edits on a single line as long as the cursor remains on that line. Once you move off the line, you can no longer use **u**.

Transition Statement — Sometimes it is convenient to execute an AIX command while still in **vi**. Let's look at how to do this next.

vi - Executing AIX Commands

\$ **vi myfile**

keystroke:

<Esc>

The following should be stocked in the employee break room:

~
~
~

:!ls

file1 file 2 snacks

[Hit return to continue]

The following should be stocked in the employee break room:

:r snacks

candy bars
soda pop
popcorn

© Copyright IBM Corporation 2008

Figure 7-12. vi - Executing AIX Commands

AU1310.0

Notes:

Introduction

Rather than ending a **vi** session to run an AIX command, only to have to return to **vi**, **vi** gives you the capability to temporarily access a shell prompt through the use of the **:!** subcommand. If you want to pull in contents of a file into an editing session, **vi** provides the **:r** subcommand.

Let's assume that there is a file in your current directory that needs to be pulled into the contents of this file. You do not want to have to rekey all the information or redirect the information after exiting the **vi** session, plus you cannot remember the name of the file. What the example shows is combining a **vi** read with a call to AIX to read the contents of **snacks** into your session.

Executing a single command from within vi

- `:!ls` will create a shell.
- All files in the current directory are listed. Press `Enter` to exit the shell and return to the `vi` session or,
- While still in command mode, issue the `:r snacks` command. The contents of the file `snacks` are read into the `vi` file. By default, it will appear after the current line.

By default the data is placed at the current line. If you have line numbering set to on, you can precede the `:r` with a line number to place the contents of the file at any point in the file.

Executing multiple commands from within vi

If you need to run a series of commands without returning to `vi` after the first command is executed, enter `:sh`. When you have run all the commands, press `<Ctrl-d>` to exit the shell and return to `vi`.

Instructor Notes:

Purpose — Discuss executing AIX commands within a **vi** session.

Details — Without using this **vi** feature you would had to have done the following:

- Exit **vi**
- Run **ls**
- Marked down the correct file name
- Reinvoke **vi**
- Move cursor to desired location
- Key in the information
- Exit **vi**, run **ls**, locate the file, and appended it to the saved **vi** file. Then you would have to reinvoke the file to yank and put the contents of the appended information in the correct location.

This is also a nice feature to dump the output of the **date** command.

Additional Information —

Transition Statement — Various default options can be used for **vi** editing sessions. We will discuss this next.

vi Options

- **Options** entered in the **vi** session change the behavior of the **vi** command

```
:set all
:set autoindent / noautoindent
:set number / nonumber
:set list / nolist
:set showmode / noshowmode
:set tabstop=x
:set ignorecase / noignorecase
:set wrapmargin=5
```

- Options can be stored in the file **\$HOME/.exrc**
- **Macros** can be written and new commands created

© Copyright IBM Corporation 2008

Figure 7-13. vi Options

AU1310.0

Notes:

Changing vi behavior

vi has many modes of operation. Some of these will affect the way text is presented, while others will make editing easier for novice users. Here are some examples:

:set all	Display all settings.
:set	Display settings different than the default.
:set autoindent	Sets autoindent on.
:set noautoindent	Turns autoindent mode off.
:set number	Enables line numbers.
:set nonumber	Turns line numbers off.
:set list	Displays non-printable characters.
:set nolist	Hides non-printable characters.

:set showmode	Shows the current mode of operation.
:set noshowmode	Hides mode of operation.
:set tabstop=4	Sets tabs to 4-character jumps.
:set ignorecase	Ignores case sensitivity.
:set noignorecase	Case-sensitive.
:set wrapmargin=5	Sets the margin for automatic word wrapping from one line to next. A value of 0 turns off word wrapping.

Options file

The file **.exrc** will be searched for in the current directory first. If found, then it will be read for settings as above.

If no **.exrc** was found in the current directory, the **HOME** directory is searched next. Finally, the built-in defaults are used.

.exrc contents are “set” options, but without the initial colon (:).

Instructor Notes:

Purpose — To show how to change the modes of the `vi` editor.

Details — The `$HOME/.exrc` file contains all the above settings with one small difference: The settings are not preceded by a colon (:).

Additional Information — There are many more modes available in `vi`. However, discussion of these is out of the scope of this class.

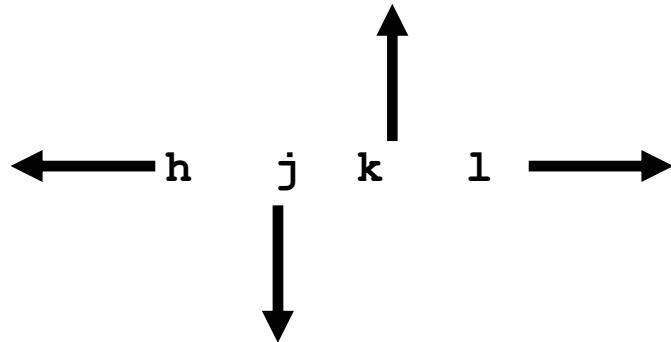
Transition Statement — The final aspect of the `vi` command is to understand its different command line forms.

Command Line Editing

- Uses same editing keys as **vi**
- Can correct **mistakes** in the current line
- Uses editor keys to **edit/reenter previous lines**

```
$ set -o vi
```

Remember:



© Copyright IBM Corporation 2008

Figure 7-14. Command Line Editing

AU1310.0

Notes:

Editing the command line

Normally, you can only edit the command line using the backspace key (**<ctrl-h>**) to back up and erase input or the interrupt key (**<ctrl-x>**). The command line editing feature of the *Korn shell* allows you to use the same keys as the **vi** editor to edit the command line and correct mistakes. Many of the editing facilities of these editors are available.

Enabling command line editing

To enable command recall, you can enter the command **set -o vi**. Once set up, previous commands can be recalled by first pressing the **Esc** key and then pressing the **k** to go “up a line.” The list of commands is read from the **.sh_history** file.

The command line editor can also be set to use the commands from the **emacs** editor. It is set the same way as **vi**: **set -o emacs**

Editing commands

You can then edit the line as you would any line of text in a **vi** editing session. Only single letter commands will work, including the search capabilities.

Executing commands

When you have edited the line, press **Enter**, and it will be processed by the shell.

Disabling command line editing

To turn off the command recall facility enter **\$ set +o vi**. Preceding any of the flags with a + (plus) rather than a - (minus) turns off the option.

Instructor Notes:

Purpose — This visual shows how to set up and use the command line editing feature in the Korn shell.

Details — The command line recall will, by default, use the same keystrokes as the **vi** editor. Previous lines may be recalled by pressing the **k** repeatedly until the desired line is reached.

Once on the line, then all the normal **vi** editing commands may be invoked; for example, **cw** to change word, and so on.

Command line editing can also be set up by setting the **EDITOR** environment variable. For example, **EDITOR=/usr/bin/vi**. This of course, can also be placed in a user's **.profile** file.

When working with some terminal emulators (such as Simon Tatham's PuTTY utility) the backspace key may need to be mapped to use **<Ctrl-h>**.

Additional Information — For a full reference of the commands available at this point, refer to the **vi** manual pages or the Web-based documentation.

Korn shell command line editing can be defined in a special file called **\$HOME/.kshrc** and will be available every time a korn shell is opened. This will be covered later in the course.

Transition Statement — Let's quickly look at different forms of the **vi** editor.

vi Editors

vi	Full-screen, full-function editor
view	Read only form of vi , changes cannot be saved unless overridden with a force (!)
vedit	Beginner's version of vi , showmode is on by default
ex, ed	Subset of vi working in line mode, can access the screen editing capabilities of vi
edit	Simple form of ex

© Copyright IBM Corporation 2008

Figure 7-15. vi Editors

AU1310.0

Notes:

Introduction

emacs is another popular UNIX editor but is not standard across all UNIX platforms.

Instructor Notes:

Purpose — To show the different forms of the `vi` editor and how these can be used.

Details —

Additional Information —

Transition Statement — Let's cover a few checkpoint questions before the exercise.

Checkpoint

1. When using the **vi** editor, what are the two modes of operation?
2. While using **vi**, how do you get to command mode?

3. Which of the following could you use to enter in text?
a
x
i
dd
4. While in command mode, pressing the u key repeatedly will "undo" all previously entered commands. True or False?
5. **vi** can be used to globally change the first occurrence of a pattern on every line with a given pattern. True or False?

© Copyright IBM Corporation 2008

Figure 7-16. Checkpoint

AU1310.0

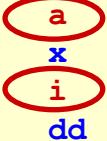
Notes:

Instructor Notes:

Purpose — To test the students' knowledge of the presented material.

Details —

Checkpoint Solutions

1. When using the **vi** editor, what are the two modes of operation? **text mode and command mode**
2. While using **vi**, how do you get to command mode? **Press the <escape> key. Remember though, the <escape> key is not a toggle. If it is pressed repeatedly, the user remains in command mode.**
3. Which of the following could you use to enter in text?

 - a
 - x
 - i
 - dd
4. While in command mode, pressing the u key repeatedly will "undo" all previously entered commands. True or False? **False. The u command will only undo the previous command.**
5. **vi** can be used to globally change the first occurrence of a pattern on every line with a given pattern. True or False? **True.**

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — Let's go to your lab systems and test out what you have learned.

Exercise: vi Editor



© Copyright IBM Corporation 2008

Figure 7-17. Exercise: vi Editor

AU1310.0

Notes:

After completing the lab, you should be able to:

- Create and edit files using the **vi** editor
- Invoke *command line editing*

Instructor Notes:

Purpose — Introduce to the lab.

Details —

Additional Information —

Transition Statement — Let's review the key points we have covered in this unit.

Unit Summary

- The **vi** command starts a **full-screen** editor.
- **vi** has two modes of operation: **text input mode** and **command mode**.
- **vi** makes a **copy** of the file you are editing in an edit buffer.
- The contents are **not changed** until you **save** the changes.
- Subcommands with the **:**, **/**, **?**, or **!** **read input** from a line displayed at the **bottom of the screen**.

© Copyright IBM Corporation 2008

Figure 7-18. Unit Summary

AU1310.0

Notes:

Instructor Notes:

Purpose — To summarize the unit's key points.

Details —

Additional Information —

Transition Statement — Time for another unit.

Unit 8. Shell Basics

Estimated Time

01:00

What This Unit Is About

This unit introduces the major functions available within shells.

What You Should Be Able to Do

After completing this unit, students should be able to:

- Use wildcards to access files with similar names
- Use redirection and pipes to control the input and output of processes
- Use line continuation in order to enter long command lines
- Group commands in order to control their execution

How You Will Check Your Progress

Accountability:

- Student activity
- Checkpoint questions
- Exercise

Unit Objectives

After completing this unit, you should be able to:

- Use **wildcards** to access files with similar names
- Use **redirection** and **pipes** to control the **input** and **output** of processes
- Use **line continuation** to enter commands that span the command line
- **Group commands** in order to control their execution

© Copyright IBM Corporation 2008

Figure 8-1. Unit Objectives

AU1310.0

Notes:

Instructor Notes:

Purpose — To introduce this unit's objectives to the students.

Details —

Additional Information —

Transition Statement — How does a user interface with the operating system?

The Shell

- Korn (ksh) or Bourne (bsh) or C (csh)
- User interface to AIX
- Command interpreter
- Enables multiple tasks
- Comprehensive programming language

© Copyright IBM Corporation 2008

Figure 8-2. The Shell

AU1310.0

Notes:

Shell features

The shell is the primary interface between the user and the operating system. The standard shell in AIX is the Korn shell.

The shell interprets user commands to start applications and use the system utilities to manage user data.

The shell enables multiple processes to be running in the background simultaneously to the foreground process with which the user is interacting.

The shell can be used as a comprehensive programming language by combining sequences of commands with the variables and flow control facilities provided by the shell.

Instructor Notes:

Purpose — This visual introduces the students to the default shell and its features.

Details — Since AIX Version 3.0 the default shell is the Korn shell and the Bourne shell exists mainly for reasons of compatibility.

Additional Information —

Transition Statement — Let's now take a look at several features of the shell.

Metacharacters and Wildcards

- **Metacharacters** are characters that the shell interprets as having a **special meaning**.

Examples:

```
< > | ; ! * ? [ ] $ \ " ' '
```

- **Wildcards** are a subset of metacharacters that are used to **search for** and **match file patterns**.

Examples:

```
* ? ! [ ] [ - ]
```

© Copyright IBM Corporation 2008

Figure 8-3. Metacharacters and Wildcards

AU1310.0

Notes:

Metacharacters

We will introduce the meaning of each of the metacharacters during the course of this unit.

Because the metacharacters have special meaning to the shell, they should not normally be used as any part of a file name.

The “-” symbol can usually be used in a filename provided it is not the first character. For example, if we had a file called `-1` then issuing the command `ls -1` would give you a long listing of the current directory because the `ls` command would think the `1` was an option rather than `-1` being a file name argument. Some AIX commands provide facilities to overcome this problem.

Available metacharacters

```
! " $ % ^ & * ( ) { } [ ] # ~ ; ' < > / ? ' \ |
```

Instructor Notes:

Purpose — This visual presents students with their first look at metacharacters. These will be explained throughout this unit and the next.

Details — All metacharacters have special meaning to the shell. The meanings of some characters vary from one shell to the next. None of these metacharacters should be used as part of a filename.

Additional Information —

Transition Statement — Let's take a look at a subset of these metacharacters called wildcards.

File Name Substitution (1 of 2)

Wildcards: * ?

- One character compare:

```
$ ls ne?
net    new
```

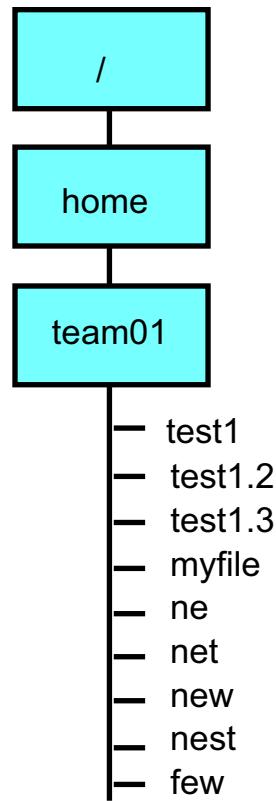
```
$ rm ?e?
few    net    new
```

- Multiple character compare:

```
$ cp n*      /tmp
ne   net      new    nest

$ qprt *w
new   few

$ echo test1*
test1  test1.2  test1.3
```



© Copyright IBM Corporation 2008

Figure 8-4. File Name Substitution (1 of 2)

AU1310.0

Notes:

Wildcards

The wildcard ? is expanded by the shell to match from any single character in a file name. The exception is that the ? will *not* match a dot “.” as the first character of a file name (for example, in a hidden file).

The wildcard * is expanded by the shell to match zero to any number of characters in a file name. The single * will be expanded to mean all files in the current directory except those beginning with a dot. Beware of the command `rm *` which could cause serious damage removing all files.

Since the shell is interpreting the wildcard metacharacters and always substitutes filenames, filename substitution is only useful for commands that use filename option values or arguments. For example, the command: `mail team*` would not expand to a list of users starting with “team.”

Instructor Notes:

Purpose — To show the use of the wildcard characters * and ?.

Details — Wildcards operate in the following manner:

The wildcard pattern is matched to file names in the given directory. If no path is provided, this will be the current working directory. The pattern must be able to match the entire file name and not just a substring of the file name. All file names which are matched are then substituted (with spaces between each file name) on the command line where the pattern was located. This is sometime referred to as “expanding” the pattern. Officially this is called “file name substitution.”

- * matches strings of any length in a file name
- ? matches a single character in a file name

It is important to understand that the shell does this “expansion” before it executes the command.

Thus, `$ ls *` might expand to `$ ls file1 dirA file3` and then the shell will execute that resulting command line.

It is good practice to execute `echo <pattern>` to see what files you are about to affect.

Discussion Items — The * in the questions will match to all the files in the current directory, so all the files will be removed. The `rm` with the `-R` option will remove the files recursively down the tree structure. So imagine if you were logged in as the root user, what kind of an effect this would have.

Another question can be asked here:

What is the difference between `echo *` and `ls *`?

(Answer: `ls` lists directory contents too, while `echo` just expands their names and works like `ls -d!` This may get confusing if you enter `echo m*` and `ls m*` when you have a subdirectory whose name starts with an `m`!)

Additional Information — Using commands with the wildcards can be dangerous. When using `rm` with wildcards, it is recommended that the interactive form of `rm` (`rm -i` or `del`) be used.

Transition Statement — Wildcard substitution can be taken further by using lists.

File Name Substitution (2 of 2)

Inclusive Lists: [] ! [-]

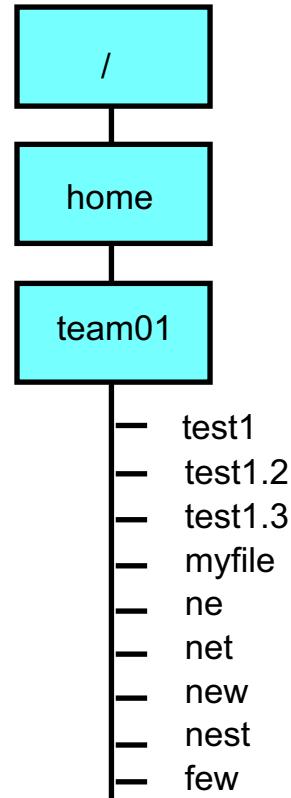
```
$ ls ne[stw]
net new

$ rm [fghjdn]e[tw]
few net new

$ ls *[1-5]
test1 test1.2 test1.3

$ qprt [!tn]*
myfile few

$ cat ?![y]*[2-5]
test1.2 test1.3
```



© Copyright IBM Corporation 2008

Figure 8-5. File Name Substitution (2 of 2)

AU1310.0

Notes:

Inclusion lists

The position held by the brackets and their contents will be matched to a single character in the file name. That character will either be a member of a list or range, or any character which is not a member of that list or range if the ! character is used at the start of the inclusion list.

Examples

The examples on the visual do the following:

- The first example will list all three letter files which begin with the letters ne and have as the last letter either s or t or w
- The second example will remove any file that begins with ONE of the characters from the first set of brackets, has the middle letter as e and ends with either t or w
- The third example will list all files that end with either 1, 2, 3, 4, or 5

- The fourth example will print all files that do not begin with the letters `t` or `n`
- The final example will display the contents of any file that has the first character as anything, the second letter must not be `y`, zero or more characters can then follow, with the last character being one from the range 2 through 5

Instructor Notes:

Purpose — Explain the metacharacters used for character substitution.

Details — Like the ?, inclusion lists match one character and one character only, and not a combination of those written inside of the [] characters.

[az] represents the letters a OR z

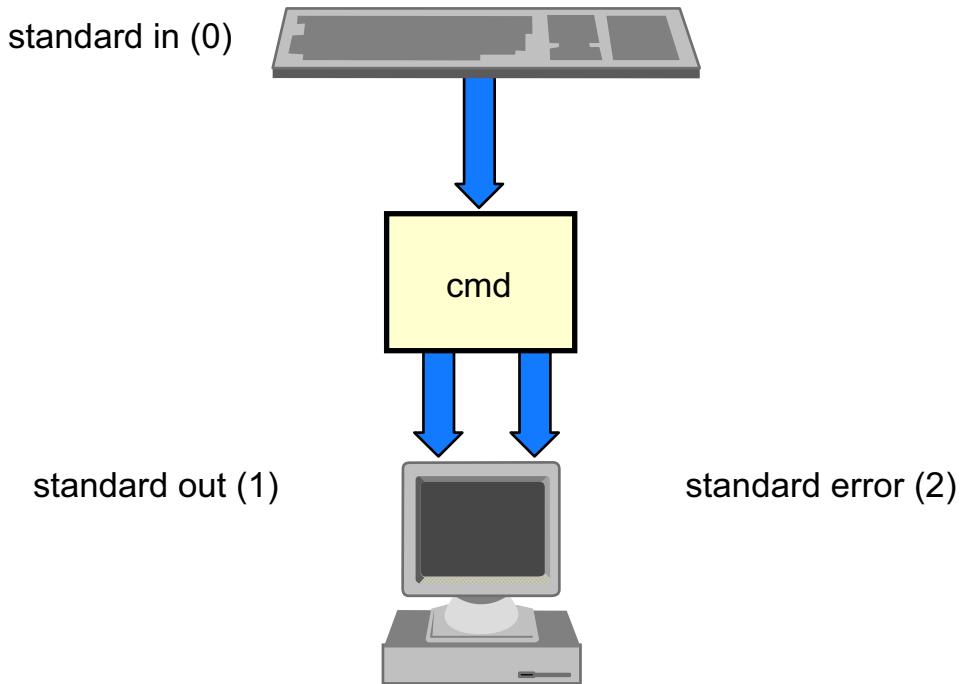
[a-z] represents the letters a through z

[!az] represents all characters EXCEPT a or z

Additional Information — When specifying a range, note that this is a range of the ASCII code value that represent the characters. For example, A is 65, Z is 90, a is 97 and z is 122. To specify that a character can be any upper or lowercase number, one would have to code [A-Za-z] because they are two discontiguous ranges. the coding of [A-z] will not work because that would also match the characters that are between 91 and 96 in the collating sequence. Characters that would be matched include: [, \,], ^, _, and `.

Transition Statement — Having seen wildcards, let us now take a look at how to redirect command input and output.

The Standard Files



© Copyright IBM Corporation 2008

Figure 8-6. The Standard Files

AU1310.0

Notes:

Command redirection

The shell is the primary interface between the user and the operating system. The standard shell in AIX is the Korn shell.

Three files are automatically opened for each process in the system. These files are referred to as *standard input*, *standard output*, and *standard error*.

When an application works with a file, it opens the file using the path to the file, but once the file is open the application uses a numerical identifier, called a file descriptor, to identify which file to read from or write to. The numbers shown on the foil are the standard file descriptors for: STDIN (0), STDOUT (1), and STDERR (2).

Standard input, sometimes abbreviated to `stdin` is where a command expects to find its input, usually the keyboard.

Standard out (`stdout`) and standard error (`stderr`) are where the command expects to put its output, usually the screen. These defaults can be changed using redirection.

Instructor Notes:

Purpose — This visual shows the students the three files that most programs expect to be open when they begin execution.

Details —

Standard Input or `STDIN`, where an application obtains its input.
This is usually the keyboard or a file.

Standard Output or `STDOUT`, where an application sends its output.
This is normally the screen or another file.

Standard Error or `STDERR`, where the application sends its error messages.
This can be a screen or the system console.

Note the file descriptor numbers and their associations. Then use this as a segue to the next slide.

Additional Information —

Transition Statement — The shell gives us the ability to override these default files with files of our choosing. Doing so is referred to as “redirection.” Let’s look at how we can be refer to each of these file descriptors when we want to redirect related standard files.

File Descriptors

- **Three descriptors** are assigned by the shell when the program starts:

Standard in:	<	0
Standard out:	>	1
Standard error:	2>	2

© Copyright IBM Corporation 2008

Figure 8-7. File Descriptors

AU1310.0

Notes:

AIX file descriptors

A *file descriptor number* is what a program uses when it needs to identify a file which it will be read from or writing to. The file descriptors will differ depending on the command or utility that is currently running. Each AIX command opens its own set of file descriptors in order to keep track of the data files, input, output, and error messages.

Special files

Remember that in AIX, not all file names refer to real data files. Some files may be *special files* which in reality are a pointer to some of the devices on the system. An example would be **/dev/tty0**.

Instructor Notes:

Purpose — Explain the file descriptors and the three standard files.

Details — Entries may refer to special device files, like a terminal or a disk drive.

Additional Information — The < symbol is, by default, shorthand for redirecting STDIN(0), while the > symbol is, by default, shorthand for redirecting STDOUT(1). Either symbol can be redefined to redirect file descriptors other than (0) or (1). For example, if my application wrote to a data file using file descriptor (5), then 5> /tmp/junk would redirect that out to /tmp/junk instead of where the application would have written it. When we use 2> to redirect STDERR (2), we are just redefining which file descriptor we are redirecting. Using 1> is the same thing as >; we are just being explicit instead of using the default.

Transition Statement — Let's see how to manipulate these file descriptors. Let's begin with the standard input descriptor.

Input Redirection

- Default standard input

```
$ mail team01
Subject: Letter
This is a letter.
<ctrl-d>
Cc:
$
```

- Redirect input from a file: <

```
$ mail team01 < letter
$
```

© Copyright IBM Corporation 2008

Figure 8-8. Input Redirection

AU1310.0

Notes:

Redirection

In the redirection example, the file **letter** can be created using an editor or word processing application. It is then used as standard input to the **mail** program rather than typing from the keyboard. This would make it much easier to format the letter or correct any typing mistakes.

With redirection and the **mail** command, you will *not* get the normal prompts for **Subject:** or **Cc:**. You must use the following syntax:

```
mail -s "My Subject" -c user1,user2 recipient
```

The symbol < tells **mail** to take input from the file instead of the keyboard.

The **mail** program handles standard out differently than other commands.

Instructor Notes:

Purpose — To demonstrate how to control STDIN for a command.

Details — Not all commands will accept redirected input. For example, `ls` or `rm` totally ignore input redirection.

Other commands expect either a filename as a parameter, or assume if it is missing that the input is from STDIN. STDIN may either be a redirected file or the terminal keyboard.

For example:

```
$ grep abc datafile.txt  
$ cat datafile.txt | grep abc
```

In the last case, input can be redirected into that command.

Additional Information —

Transition Statement — Let us now look at managing the output of a command.

Output Redirection

- Default standard output:

```
$ ls
file1 file2 file3
```

- Redirect output from a file: >

```
$ ls > ls.out
$
```

- Redirecting and appending output to a file: >>

```
$ who >> whos.there
$
```

© Copyright IBM Corporation 2008

Figure 8-9. Output Redirection

AU1310.0

Notes:

Standard output redirection

Redirection allows standard output to go to somewhere other than the screen (default). In the example, standard output has been redirected to go to the file named **ls.out**.

The file descriptor table in this example will hold the following values:

0 (unchanged)	STDIN
1 (changed)	ls.out
2 (unchanged)	STDERR

Using ordinary redirection can overwrite an existing file. To avoid this, use **>>** (no space between them) to *append* the output to an existing file.

The file descriptors for the *append* example will be as follows:

0 (unchanged)	STDIN
1 (changed)	whos.there
2 (unchanged)	STDERR

Instructor Notes:

Purpose — To demonstrate how to control STDOUT.

Details — Output redirection can be used to store the result of a command for future reference.

Not all commands produce output that can be redirected. For example, printing commands write to the printer.

Warning: In the `ls` example, if the file `ls.out` exists, it will be overwritten.

Note: In the `who` example, if the file does not exist it will be created.

Discussion Items — Ask the students the following:

Boardwork: `$ banner hello > /dev/tty1`

What will this command do?

Answer: Output the string `hello` on the terminal `tty1` (provided `tty1` has not set `msg_n`).

Additional Information —

Transition Statement — Let's take another look at a use for output redirection, by using the `cat` command.

Creating a File with cat

- While normally used to list the contents of files, using cat with redirection can be used to create a file:

```
$ ls
letter acctfile file1

$ cat file1
This is a test file.
The file has 2 lines.
$
```

- Using redirection:

```
$ cat > newfile
This is line 1 of the file.
This is the 2nd line.
And the last.
<ctrl-d>

$ ls
letter acctfile file1 newfile
```

© Copyright IBM Corporation 2008

Figure 8-10. Creating a File with `cat`

AU1310.0

Notes:

Other uses for cat

You already learned in an earlier activity that you can create files with the `cat` command.

For the `cat > newfile` example, the file descriptors will hold the following information:

0 (unchanged)	STDIN
1 (changed)	newfile
2 (unchanged)	STDERR

Instructor Notes:

Purpose — Demonstrate another use for output redirection.

Details — When using `cat` in this way, be careful not to use `<ctrl-c>` as this will cancel the command. `<Ctrl-d>` stands for end of data.

The special device file `/dev/null` is often used to discard unwanted output from a command. It can also be used to create a zero-byte file.

For example:

```
$ cat /dev/null > newfile  
$ cat file > /dev/null
```

Zero-byte files can also be created without the `cat` command being involved.

For example:

```
>newfile
```

This creates a file called `newfile`. However, if `newfile` exists then its contents will be removed and the file will become zero bytes long.

Discussion Items — What is the effect of entering:

- a. `$ cat <enter>`
- b. `$ cat file1 newfile > file.out`

Answer:

- a. The system will wait for you to type something, then echo it back once the Enter key is pressed. This will continue until a `<ctrl-c>` is entered.
- b. The contents of both `file1` and `newfile` will be placed into `file.out`.

Additional Information —

Transition Statement — Let's do an activity, to review shell basics.

Activity: Review Shell Basics

1. Which files are listed when the following commands are executed?

```
$ ls /home/team01/*.?
```

```
$ ls /tmp/[a-zA-Z]*.[0-9]
```

2. True or False: The command "ls *" lists all files in a directory.

3. Write down the **file descriptors** for the following command:

```
$ wc -l < file1 > /tmp/lines
```

Standard input:

Standard output:

Standard error:

4. You want to append file **testfile1** to file **report99**. Which command is correct?

- cat report99 < testfile1
- cat testfile1 > report99
- cat testfile1 report99
- cat testfile1 >> report99

© Copyright IBM Corporation 2008

Figure 8-11. Activity: Review Shell Basics

AU1310.0

Notes:

Activity

Please answer the questions in the visual.

Instructor Notes:

Purpose — To test the students' knowledge of the material in this unit on shell basics.

Details — Give the students some time to answer the questions. Review the answers afterwards together with the students.

Additional Information —

Transition Statement — Let us now take a look at the STDERR file descriptor.

Error Redirection

- Default standard error:

```
$ cat filea fileb
This is output from filea.
cat: cannot open fileb
```

- Redirecting error output to a file: 2> (To append: 2>>)

```
$ cat filea fileb 2> errfile
This is output from filea

$ cat errfile
cat: cannot open fileb

$ cat filea fileb 2> /dev/null
This is output from filea
```

© Copyright IBM Corporation 2008

Figure 8-12. Error Redirection

AU1310.0

Notes:

STDERR redirection

Error messages from commands would normally be sent to the screen. This can be changed by redirecting the STDERR to a file.

When redirecting output, there can be no spaces between the 2 and the >.

The file descriptor table for the first error redirection example will contain the following:

0 (unchanged)	STDIN
1 (unchanged)	STDOUT
2 (changed)	errfile

and for the second:

0 (unchanged)	STDIN
1 (unchanged)	STDOUT
2 (changed)	/dev/null

Special file

The special file **/dev/null** is a bottomless pit where you can redirect unwanted data. All data sent there is just thrown away.

/dev/null is a special file. It has a unique property as it is always empty. It is commonly referred to as the bit bucket.

Instructor Notes:

Purpose — Show how standard error is manipulated.

Details — In some situations, for example, in a script or shell program, error messages may need to be collected for future reference.

Additional Information — When using error redirection, emphasize that the syntax is `2>` and not `2 >!` In other words there must be no intervening space between the file descriptor number and the redirection symbol.

Transition Statement — Redirection can get rather complicated by entering all three on the same command line.

Combined Redirection

- **Combined redirects:**

```
$ command > outfile 2> errfile < infile  
$ command >> appendfile 2>> errfile < infile
```

- **Association examples:**

Redirect standard error to standard out:

```
$ command > outfile 2>&1
```

CAUTION: This is NOT the same as above

```
$ command 2>&1 > outfile
```

© Copyright IBM Corporation 2008

Figure 8-13. Combined Redirection

AU1310.0

Notes:

Combined redirection example

One may redirect multiple file descriptors on a single command. Normally the order in which you do the redirections is not important, unless you are using association.

Association example

In the first example, file descriptor 1 is associated with the file specified, **outfile**. Then the example associates descriptor 2 with the file associated with file descriptor 1, **outfile**.

If the order of the redirection is reversed as in the second example, then file descriptor 2 would be associated with the terminal (standard out) and file descriptor 1 would be associated with the file specified **outfile**.

Order of redirection in associations

With the association examples, the order in which redirections are specified is significant. For association, here is an example of **ls**:

```
ls -l / > ./list.file 2>&1
```

0 (unchanged)	STDIN
1 (changed)	./list.file
2 (changed)	./list.file

And here is an example of how *not* to do association:

```
ls -l / 2>&1 > ./list.file
```

0 (unchanged)	STDIN
1 (changed)	./list.file
2 (unchanged)	STDOUT

In the second association example, the errors are redirected to the same place as standard out. But standard out at this point has not been redirected yet, so the default value will be used which is the screen. So, the error messages will be redirected to the screen. Remember that by default error messages are sent to the screen.

Instructor Notes:

Purpose — To show combining redirection on the same command line.

Details — Ensure that the last example is fully explained. Point out that although in the first two examples the ordering of the specified redirections does not matter, when association is used, care must be taken to specify the redirections in the correct order.

Additional Information —

Transition Statement — Redirection forces the output (or input) of a command to be placed to a file, or somewhere other than the default for the command.

The next visual takes this one step further.

Pipes

A sequence of one or more commands separated by a vertical bar "|" is called a **pipe**. The **standard output** of each command becomes the **standard input** of the next command.

```
$ who | wc -l
4
```

This is the same as:

```
$ who > tempfile
$ wc -l tempfile
4      tempfile
$ rm tempfile
```

© Copyright IBM Corporation 2008

Figure 8-14. Pipes

AU1310.0

Notes:

Command pipes

Two or more commands can be separated by a pipe on a single command line. The requirement is that any command to the left of a pipe must send output to standard output. Any command to the right of the pipe must take its input from standard input.

The example on the visual shows that the output of **who** is passed as input to **wc -l**, which gives us the number of active users on the system.

Instructor Notes:

Purpose — To show how the output from one command may become the input for another.

Details — All elements of a pipe execute in parallel. If one of the commands in a pipe fails, then the whole pipe will fail.

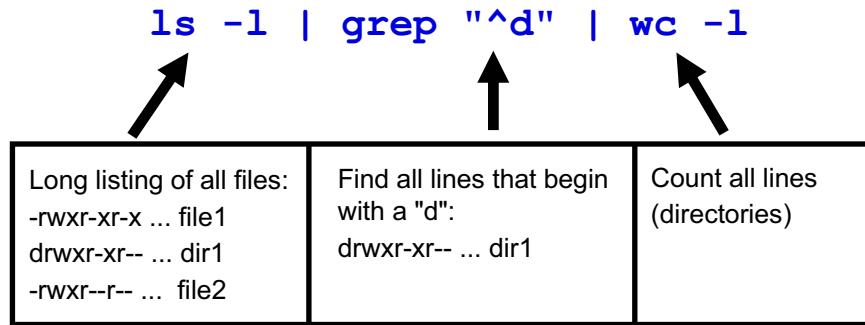
Additional Information —

Transition Statement — There are other uses for pipelines.

Filters

A **filter** is a command that **reads from standard in, transforms the input** in some way, and **writes to standard out**.

Example:



© Copyright IBM Corporation 2008

Figure 8-15. Filters

AU1310.0

Notes:

Filter uses

A command is referred to as a filter if it can read its input from standard input, alter it in some way, and write its output to standard output. A filter can be used as an intermediate command between pipes.

A filter is commonly used with a string of piped commands, as in the example above. The **ls -l** command lists all the files in the current directory and then pipes this information to the **grep** command. The **grep** command will be covered in more detail later in the course, but in this example, the **grep** command is used to find all lines beginning with a d (directories). The output of the **grep** command is then piped to the **wc -l** command. The result is that the command is counting the number of directories. In this example, the **grep** command is acting as a filter.

Instructor Notes:

Purpose — Explain the difference between a filter and a pipe.

Details — As the students become more advanced in the course, or in their own knowledge, other examples of filters may be used.

For the visual example, keep to the explanation given in the student notes. Since we have not yet covered `grep` or regular expressions, avoid being tricked into teaching those topics early.

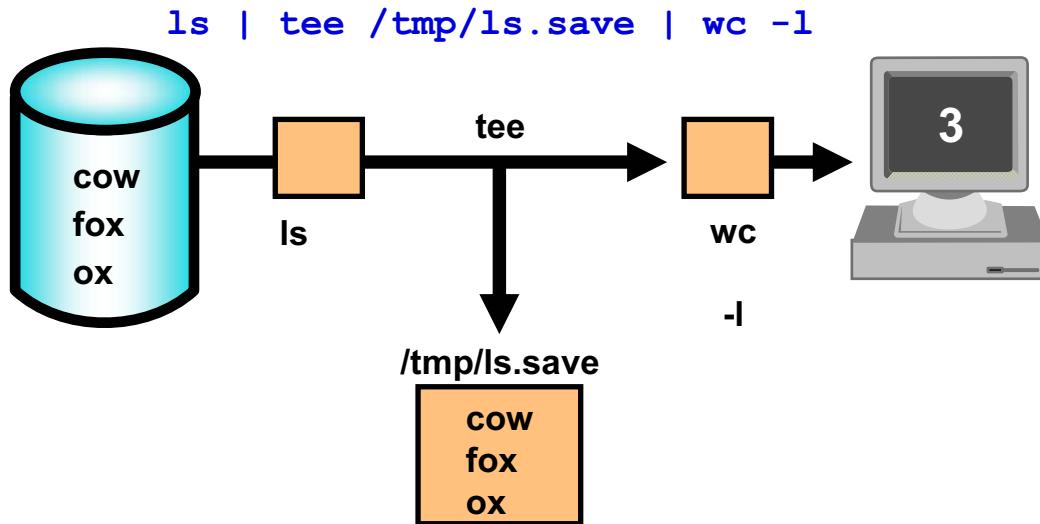
Emphasize that there are commands which can not act as filters because they do not read from STDIN. An example of such a command is `ls`.

Additional Information —

Transition Statement — If you want to preserve the original data when using pipes as well as the result, you will need a `tee`.

Split Outputs

The **tee** command reads standard input and sends the data to both standard output and a file.



© Copyright IBM Corporation 2008

Figure 8-16. Split Outputs

AU1310.0

Notes:

tee command

The **tee** command is a filter that can be used to capture a snapshot of information going through a pipe. **tee** puts a copy of the data in a file as well as passing it to standard output to be used by the next command. **tee** does not alter the data.

Instructor Notes:

Purpose — To show how information may be tapped from a pipeline.

Details — The **tee** command takes its input and routes to two places:

1. The terminal by default, unless you pipe output to another command as shown here.
2. File of your choice.

In our example, the output of the **tee** command is placed in the **/tmp/ls.save** file. This prevents the **/tmp/ls.save** file from being counted by the **wc -l** command.

Additional Information —

Transition Statement — Another metacharacter we will look at is the command grouping character.

Command Grouping

Multiple commands can be entered on the same line, separated by a semi-colon ":":

```
$ ls -R > outfile ; exit
```

is equivalent to entering:

```
$ ls -R > outfile  
$ exit
```

© Copyright IBM Corporation 2008

Figure 8-17. Command Grouping

AU1310.0

Notes:

Grouping commands

Placing multiple commands separated by a ";" on a single line produces the same result as entering each command on a separate command line.

Instructor Notes:

Purpose — To show how the shell can deal with multiple commands on the same line.

Details — Reason for doing this: Some commands can take a while to execute, such as the `ls -R` command. This allows the second command to execute once the first command completes.

Additional Information — Instead of the simple commands in our example, we could have entire pipelines of commands with each pipeline separated by a semicolon.

Transition Statement — We have just seen how we can group together a number of commands on the one line. But what if we only have one command which will not fit onto the one line. How can we overcome this problem? Let's look at line continuation.

Line Continuation

The backslash (\) followed by a new line character can be used to **continue a command** on a **separate line**.

A **secondary prompt character** ">" is issued by the shell to **indicate line continuation**.

```
$ cat /home/mydir/mysubdir/mydata \
> /home/yourdir/yoursubdir/yourdata
```

© Copyright IBM Corporation 2008

Figure 8-18. Line Continuation

AU1310.0

Notes:

Continuing the command line in shell

The **Enter** key preceded by a backslash (\<Enter>) can be used to continue a command on a separate line.

Do not confuse the continuation prompt > with the redirection character >. The secondary prompt will not form part of the completed command line. If you require a redirection character you must type it explicitly.

Instructor Notes:

Purpose — To show how line continuation can be used within the shell.

Details — At this point, do not explain in detail that the backslash is escaping the special meaning of the next character, which in this case is the **Enter** key.

The character following the backslash must be the **Enter** key.

The prompt changes from the \$ to the > which is commonly known as the secondary prompt. This is the prompt which indicates that the shell is expecting more input from the user.

Additional Information — The value of the secondary prompt is held in a variable called PS2 - which could be set to something like Carry on typing \$ to aid the user.

Also note that the secondary prompt is often seen when there is a mismatch of quotation marks; that is, single quotes ('') and the double quotes (""), at the shell prompt. To rectify from this problem, either match the missing quote, or press <**Ctrl-c**>.

Transition Statement — Let's cover a few checkpoint questions before we go to the lab exercises.

Checkpoint (1 of 2)

1. What will the following command match?

```
$ ls ???[!a-z]*[0-9]t
```

2. For questions 2-4, indicate where the standard input, standard output and standard error will go.

```
$ cat file1
```

standard input (0):

standard output (1):

standard error (2):

3. \$ mail tim < letter

standard input (0):

standard output (1):

standard error (2):

© Copyright IBM Corporation 2008

Figure 8-19. Checkpoint (1 of 2)

AU1310.0

Notes:

Instructor Notes:

Purpose — To test the students' knowledge of the unit's material.

Details —

Checkpoint Solutions (1 of 2)

1. What will the following command match?

`$ ls ???[!a-z]*[0-9]t`

This will list all the files that match the following criteria:

- the first three characters can be anything
- the fourth character must not be from the range a to z
- zero or more characters can follow
- the second-last character must be from the range 0 to 9
- the last character must be a t.

2. For questions 2-4, indicate where the standard input, standard output and standard error will go.

`$ cat file1`

standard input (0):	keyboard
standard output (1):	screen
standard error (2):	screen

3. `$ mail tim < letter`

standard input (0):	letter
standard output (1):	screen
standard error (2):	screen

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — A few more checkpoint questions on the next visual.

Checkpoint (2 of 2)

4. `$ cat .profile > newprofile 2>1`
standard input (0):
standard output (1):
standard error (2):

For questions 5, 6, and 7, create command lines to display the content of **filea** using **cat** and then perform the following:

5. Place the output of the command in **fileb** and the errors in **filec**.
6. Place the output of the command in **fileb** and associate any errors with the output in **fileb**.
7. Place the output in **fileb** and discard any error messages.
(Do not display or store error messages.)

© Copyright IBM Corporation 2008

Figure 8-20. Checkpoint (2 of 2)

AU1310.0

Notes:

Instructor Notes:

Purpose — To test the students' knowledge of the unit's material.

Details —

Checkpoint Solutions (2 of 2)

4. `$ cat .profile > newprofile 2>1`
standard input (0): **keyboard**
standard output (1): **newprofile**
standard error (2): **a file named 1**

For questions 5, 6, and 7, create command lines to display the content of **filea** using **cat** and then perform the following:

5. Place the output of the command in **fileb** and the errors in **filec**.
`$ cat filea > fileb 2> filec`
6. Place the output of the command in **fileb** and associate any errors with the output in **fileb**.
`$ cat filea > fileb 2>&1`
7. Place the output in **fileb** and discard any error messages. (Do not display or store error messages.)
`$ cat filea > fileb 2> /dev/null`

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — Let's do an exercise.

Exercise: Shell Basics



© Copyright IBM Corporation 2008

Figure 8-21. Exercise: Shell Basics

AU1310.0

Notes:

After completing this exercise, you will be able to:

- Use *wildcards* for file name expansion
- *Redirect* standard input, standard output, and standard error
- Use *pipes*, *command grouping*, and *line continuation*

Instructor Notes:

Purpose — To introduce the exercise to the students.

Details — Describe what students will learn in the lab.

Additional Information —

Transition Statement — Let' summarize this unit's key points before moving on to a new unit.

Unit Summary

- Wildcards, * and ?, provide a convenient way for specifying multiple files or directory names.
- The wildcard notation [] is like using the ? but it allows you to choose specific characters to be matched.
- Three files automatically opened for redirection are standard in, standard out, and standard error.
- I/O redirection alters the default input source or output destination of a command.
- A pipe passes the output of one command directly to the input of another command.
- A filter takes input from standard in, transforms it, and sends the output to standard out.
- tee takes input and routes it to two places, standard out and a file.

© Copyright IBM Corporation 2008

Figure 8-22. Unit Summary

AU1310.0

Notes:

Instructor Notes:

Purpose — To summarize the unit's key points.

Details —

Additional Information —

Transition Statement — Let's move on to a new unit.

Unit 9. Using Shell Variables

Estimated Time

00:40

What This Unit Is About

This unit introduces variables and quoting metacharacters.

What You Should Be Able to Do

After completing this unit, students should be able to:

- List variables that define your environment
- Set, reference, and delete variable values
- Define the use of the following quoting metacharacters: double quotes (") single quotes (') and the backslash (\)
- Perform command substitution

How You Will Check Your Progress

Accountability:

- Checkpoint questions
- Exercise

Unit Objectives

After completing this unit, you should be able to:

- List variables that define your environment
- Set, reference, and delete variable issues
- Define the use of the following quoting metacharacters:
 - Double quotes "
 - Single quotes '
 - Backslash \
- Perform command substitution

© Copyright IBM Corporation 2008

Figure 9-1. Unit Objectives

AU1310.0

Notes:

Instructor Notes:

Purpose — To introduced this unit's objectives to the students.

Details —

Additional Information —

Transition Statement — Let's start this unit off with a look at shell variables.

Shell Variables

- **Variables** represent data whose value may change
- **Shell variables** define your **environment**:
 - **HOME** Directory (such as `/home/team01`)
 - **TERMinal** Type (such as `ibm3151`)
 - Executable program search **PATH** (such as `/bin:/usr/bin:/etc..`)
- **Additional variables** can be defined

© Copyright IBM Corporation 2008

Figure 9-2. Shell Variables

AU1310.0

Notes:

Shell variable conventions

All shell variable names are case-sensitive. For example, `HOME` and `home` are not the same.

As a convention, uppercase names are used for the standard variables set by the system and lowercase names are used for the variables set by the user.

Additional variables

In addition to the variables discussed above, there are other variables that the shell maintains which will be discussed later.

Instructor Notes:

Purpose — To explain what variables are and why they exist.

Details — Variables are used extensively in AIX. The user's environment is conditioned by the variables which are set.

There are two classifications of variables:

- Those significant to the shell or other AIX commands (system-defined)
- User-defined variables

AIX variables are, by convention, in uppercase.

User-defined variables are, again by convention, in lowercase, but there is no rule about this.

Additional Information —

Transition Statement — Having defined what a variable is and what types of variables there are, let's look at the variables that are set.

Listing Variable Settings

```
$ set  
HOME=/home/team01  
PATH=/bin:/usr/bin:/etc:/home/team01/bin:.  
PS1=$  
PS2=>  
SHELL=/usr/bin/ksh  
TERM=ibm3151  
xy=day  
$ -
```

© Copyright IBM Corporation 2008

Figure 9-3. Listing Variable Settings

AU1310.0

Notes:

Setting variables

The **set** command displays the names and values of all shell variables. The **set** command is a built-in command of the shell, and therefore gives a different output depending on the shell being run, for instance a Bourne or a Korn shell.

Instructor Notes:

Purpose — To show the student the variables currently set in their environment.

Details — The `set` command just reports the variables set in this shell. If programs are invoked, they will not necessarily be able to pick the current settings up because the variables might not be exported.

The final line of the `set` output will display an underscore. The value of the underscore is not consistent, but often refers to the last word in the last command. Most users do not need to be concerned with this.

Additional Information — There is an AIX command, `/usr/bin/env`, which also will report your current environment with the variables and values listed in the same format as the `set` shell command output.

Transition Statement — Now, let's see how to create and access these variables.

Setting and Referencing Shell Variables

1. To assign a value to a shell variable:

```
name=value
```

2. To reference a variable, prefix its name with a \$ sign:

```
$ xy="hello world"
$ echo $xy
hello world
```

3. To delete a variable, use the **unset** command:

```
$ unset xy
$ echo $xy
```

```
$
```

© Copyright IBM Corporation 2008

Figure 9-4. Setting and Referencing Shell Variables

AU1310.0

Notes:

Variable contents

Variables can hold any type of data, like integer numbers, single words, text strings or even complex numbers.

It is up to the application referencing the variable to decide what to do with the contents of that variable.

The contents of the system-defined variables is fairly static, for example, the HOME variable can only contain a path to a directory file and not, for instance, a file.

Listing variables

To set a variable, use the = with NO SPACES on either side. Once the variable has been set, to refer to the value of that variable precede the variable name with a \$. There must be NO SPACE between the \$ and the variable name.

The **echo** command displays the string of text to standard out (by default to the screen).

Instructor Notes:

Purpose — This visual shows how shell variables are created, changed, and their values displayed.

Details — Remember to point out that the dollar (\$) symbol is a metacharacter that is used to prefix a variable name.

When setting variables to contain strings of text, point out that these need to be quoted.

Additional Information — While the main point of this visual is to describe the listing of variables, this is a good time to introduce the students to the meanings of some of the standard environment variables:

- a. `HOME` - holds the full path to the user's home directory.
- b. `PATH` - holds a colon delimited list of directories to search, in sequence, when a user does not provide a path to the command. If the user provides a path to the command the shell only looks in the specified directory and does not use the PATH variable.
- c. `PS1` - holds the value of the primary prompt string issued by the shell.
- d. `PS2` - holds the value of the secondary prompt string issued by the shell for line continuations.
- e. `SHELL` - hold the full path to the user's default shell.
- f. `TERM` - hold the terminal type used to identify the how to interact with the user's terminal.

Transition Statement — Now let's see a few examples of using shell variables.

Shell Variables Example

```
$ xy=day
$ echo $xy
day

$ echo Tomorrow is Tues$xy
Tomorrow is Tuesday

$ echo There will be a $xylong meeting
There will be a meeting

$ echo There will be a ${xy}long meeting
There will be a daylong meeting
```

© Copyright IBM Corporation 2008

Figure 9-5. Shell Variables Example

AU1310.0

Notes:

Examples

Notice there need not be a space BEFORE the \$ of the variable in order for the shell to do variable substitution. Note, though, what happened when there was no space AFTER the variable name. The shell searched for a variable whose name was `xylong`, which did not exist. When a variable that has not been defined is referenced, the user does not get an error. Rather a null string is returned.

To eliminate the need for a space after the variable name, the curly braces { } are used. Note that the \$ is OUTSIDE of the braces.

Instructor Notes:

Purpose — Show the students how shell variables are set and referenced and point out a potential pitfall, if curly braces are not used.

Details — A reminder about setting variables with strings of text; they must be quoted for them to work.

Additional Information —

Transition Statement — So far we have seen how to place a value into a variable. We now will see how to place the result of a command into a variable.

Command Substitution

Variable = `Output from a Command`

```
$ date
Wed 11 Jul 11:38:39 2003
$ now=$($date)           (or now=`$date`)
$ echo $now
Wed 11 Jul 11:38:39 2003
$ HOST=$($hostname)      (or HOST=`$hostname`)
$ echo $HOST
sys1
$ echo "Today is `date` and `who | wc -l` users \
> are logged in"
Today is Wed 11 Jul 11:45:27 2003 and 4 users are logged in
```

© Copyright IBM Corporation 2008

Figure 9-6. Command Substitution

AU1310.0

Notes:

Setting variables with command output

A variable can be set to the output of some command or group of commands by using the back quotes (also referred to as grave accents). They should not be mistaken for single quotes. In the examples the output of the **date** and **who** commands are stored in variables.

The back quotes are supported by the bourne shell, C shell and Korn shell. The use of **\$ (command)** is specific to the Korn shell.

When the shell sees a command substitution string on a command line, it will execute the enclosed command and will then substitute the entire command substitution string with the standard output of that command. After completing the substitution(s), the shell will then execute the resulting line.

Instructor Notes:

Purpose — This visual shows how a variable can be used to store the result of a command.

Details — The metacharacters introduced here are:

'command' the back quotes (`) character, used in pairs to surround the command whose output we would like to capture. (This back quote can be found to the left of the “1” key on most of the U.S. classroom keyboards.)

\$(command) Korn shell specific syntax

The command is carried out only once. The result is then available until the variable changes or is removed.

The back quotes are supported by the Bourne and C shells. It is also supported by the Korn shell for backward compatibility reasons. The use of \$(command) is specific to the Korn shell. For this reason, be sure to point out this syntax to the students.

Additional Information — Command substitution is not restricted to variable assignments. You could, instead, use command substitution to provide an argument to a command. For example:

- 1) `$ file `whence chtz`` This would tell me if the `chtz` command is an object module or a script I can read
- 2) `$ more `whence chtz`` I can then read the script without typing in the path to that script

Transition Statement — Up to this point we have seen how special metacharacters are used in the shell to provide for file name substitution, command substitution, redirection, and command grouping. What if we wish to use a character as text instead of having the shell act upon its special metacharacter meaning?

Quoting Metacharacters

<p>' ' Single Quotes:</p> <pre>\$ echo '\$HOME'</pre> <p>\$HOME</p>	Ignores all metacharacters between the quotes
<p>" " Double Quotes:</p> <pre>\$ echo "\$HOME"</pre> <p>/home/team01</p>	Ignores all metacharacters except for dollar \$, backquotes ` and backslash \
<p>\ Backslash:</p> <pre>\$ echo \\$HOME</pre> <p>\$HOME</p>	Ignores the special meaning of the following character

© Copyright IBM Corporation 2008

Figure 9-7. Quoting Metacharacters

AU1310.0

Notes:

The use of quotes

Quoting is used to override the shell's interpretation of special characters. Quotes allow a metacharacter to be interpreted literally instead of expanded.

You can use the backslash \ to stop the shell from interpreting one of the quoted characters.

For example:

```
$ echo "This is a double quote \""
This is a double quote "
```

Instructor Notes:

Purpose — This visual deals with the problem of how metacharacters can be disabled temporarily, within, for example, a string of text.

Details — Remember to emphasize that the quotes HAVE to go in pairs.

If quotes are mismatched, or there is an uneven number of quotes, then the prompt changes to a > character.

Recap on the backslash character (\) and that it disables the meaning of the next character which in the case of the line continuation, escapes the special meaning of the **Enter** key.

Discussion Items — Once you have gone through all the examples, ask what the following will produce:

```
$ echo \\$HOME
```

Answer:

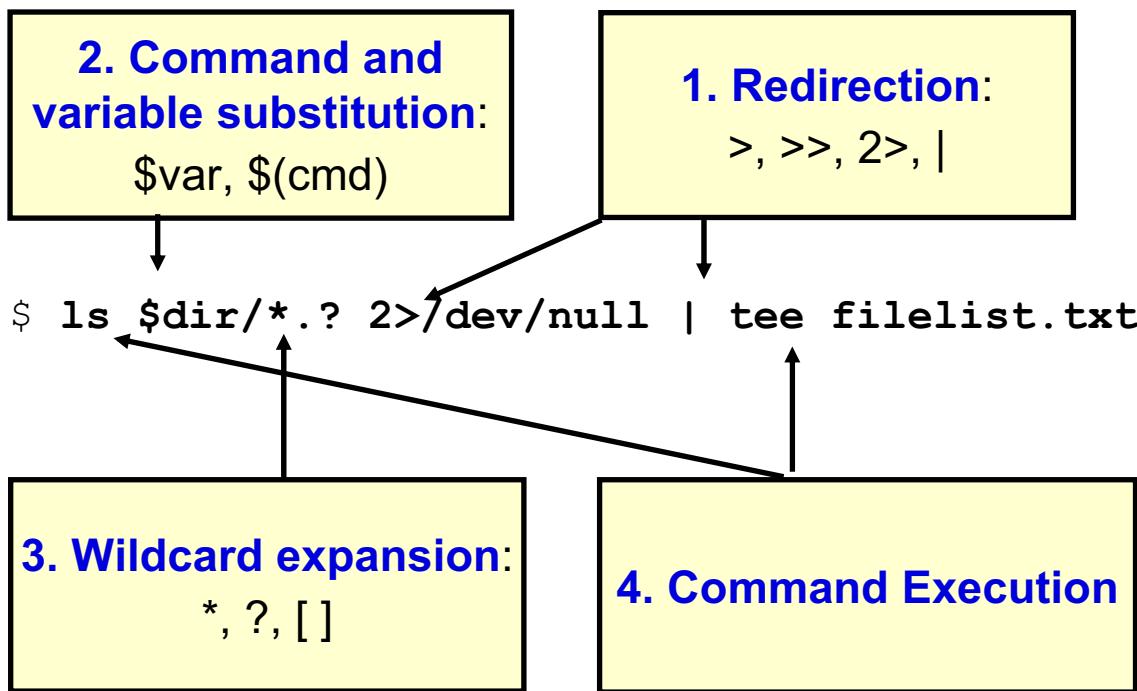
```
\/home/team01
```

Additional Information — There is a pattern to the exceptions when using double quotes. Within double quotes, we can still do:

1. File Name Substitution \${pattern}
2. Command Substitution \$(pattern)
3. Selectively use double quotes ("), back quote (`), dollar sign (\$), or backslash (\) as text.

Transition Statement — Let's recap on all the command line parsing options that we have covered in the last two units.

Command Line Parsing



© Copyright IBM Corporation 2008

Figure 9-8. Command Line Parsing

AU1310.0

Notes:

Command line parsing options

When the shell parses a command line, it is breaking the line into a series of words. One of these words determines which command will execute. Other words are information passed to the commands such as file names and options. Some of the words are instructions to the shell, like redirection.

Understand from this that the shell does a lot of “stuff” with a command line before the command ever gets to execute. The order in which the shell reads and processes a command is done from left to right. In logical order, the shell looks for redirection, command and variable substitution, wildcard expansion. The command is then executed.

Instructor Notes:

Purpose — Identify all the command line parsing options considered in the last two units.

Discussion Items — This visual simply pulls together ideas that have been discussed in the last two units.

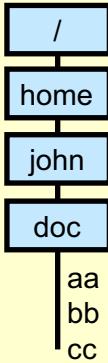
Additional Information —

Transition Statement — Before going to lab, a few checkpoint questions.

Checkpoint (1 of 2)

1. What are the results of the following commands? (Assume: the home directory is **/home/john**, the current directory is **/home/john/doc**, and it contains files **aa**, **bb** and **cc**.)

```
$ pwd  
/home/john/doc
```



2. \$ echo "Home directory is \$HOME"

3. \$ echo 'Home directory is \$HOME'

© Copyright IBM Corporation 2008

Figure 9-9. Checkpoint (1 of 2)

AU1310.0

Notes:

Instructor Notes:

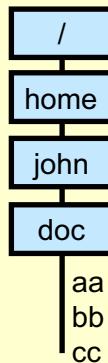
Purpose — To test the students' knowledge of the presented material.

Details —

Checkpoint Solutions (1 of 2)

- What are the results of the following commands? (Assume: the home directory is **/home/john**, the current directory is **/home/john/doc**, and it contains files **aa**, **bb** and **cc**.)

```
$ pwd
/home/john/doc
```



- \$ echo "Home directory is \$HOME"

Home directory is **/home/john**

- \$ echo 'Home directory is \$HOME'

Home directory in **\$HOME**

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — A few more questions...

Checkpoint (2 of 2)

4. \$ echo "Current directory is `pwd`"

5. \$ echo "Current directory is \$(pwd)"

6. \$ echo "Files in this directory are *"

7. \$ echo * \$HOME

8. \$ echo *

© Copyright IBM Corporation 2008

Figure 9-10. Checkpoint (2 of 2)

AU1310.0

Notes:

Instructor Notes:

Purpose — To test the students' knowledge of the presented material.

Details —

Checkpoint Solutions (2 of 2)

4. \$ echo "Current directory is `pwd`"
Current directory is /home/john/doc
5. \$ echo "Current directory is \$(pwd)"
Current directory is /home/john/doc
6. \$ echo "Files in this directory are *"
File in this directory are *
7. \$ echo * \$HOME
aa bb cc /home/john
8. \$ echo *

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — Let's do a lab exercise.

Exercise: Using Shell Variables



© Copyright IBM Corporation 2008

Figure 9-11. Exercise: Using Shell Variables

AU1310.0

Notes:

After completing the exercise, you will be able to:

- List shell built-in variables
- Use variable and command substitution
- Use quoting metacharacters

Instructor Notes:

Purpose — To introduce the next exercise to the students.

Details — Describe what students will learn in the lab.

Additional Information —

Transition Statement — Let's wrap up the unit with a quick summary review.

Unit Summary

- The shell has **variables** which define your **environment** and lets you define variables of your own.
- Variables can be set to a **value** which can then be referenced and used within scripts.
- The following **quoting metacharacters** have been discussed:
 - Double quote (" ")
 - Single quote (' ')
 - Backslash (\)
- Perform **command substitution** using either backquotes (` `) or \$(command).

© Copyright IBM Corporation 2008

Figure 9-12. Unit Summary

AU1310.0

Notes:

Instructor Notes:

Purpose — To wrap up the unit with a summary.

Details —

Additional Information —

Transition Statement — Let's move on to the next unit.

Unit 10. Processes

Estimated Time

00:50

What This Unit Is About

This unit introduces processes, their environment, and how processes are created. The discussion includes shell scripts and how they are invoked.

What You Should Be Able to Do

After completing this unit, students should be able to:

- Define an AIX process
- Describe the relationship between parent and child processes
- Create and invoke shell scripts

How You Will Check Your Progress

Accountability:

- Student activity
- Checkpoint questions
- Exercise

Unit Objectives

After completing this unit, you should be able to:

- Define an AIX process
- Describe the relationship between parent and child processes
- Create and invoke shell scripts

© Copyright IBM Corporation 2008

Figure 10-1. Unit Objectives

AU1310.0

Notes:

Instructor Notes:

Purpose — To introduce the objectives for this lecture.

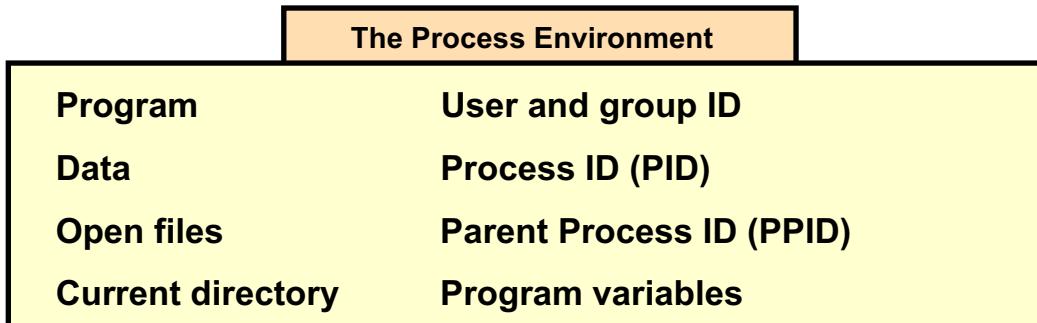
Details — Here is an outline of objectives for this unit. It describes how AIX jobs or commands work and how you can interact with them.

Additional Information —

Transition Statement — Let's begin by defining a process.

What Is a Process?

- Each program runs in a process:



- The variable `$$` shows the process ID of the current shell:

```
$ echo $$
```

4712

- The `ps` command shows the running processes:

```
$ ps -u team01
```

© Copyright IBM Corporation 2008

Figure 10-2. What Is a Process?

AU1310.0

Notes:

AIX processes

A program or a command that is actually running on a system is referred to as a process. AIX can run a number of different processes at the same time as well as many occurrences of a program (such as `vi`) existing simultaneously in the system.

The process ID (PID) is extracted from a process table.

In a shell environment, the process ID is stored in the variable `$$`.

Listing processes

To identify the running processes, execute the command `ps`, which will be covered later in this course. For example, `ps -u team01` shows all running processes from user **team01**.

Instructor Notes:

Purpose — Define a process, what it is, its environment, and the type of information it contains. The next few visuals will describe this.

Details — A process can be regarded as a running command, for example `ls`. While the process is running (active), it will be known to AIX.

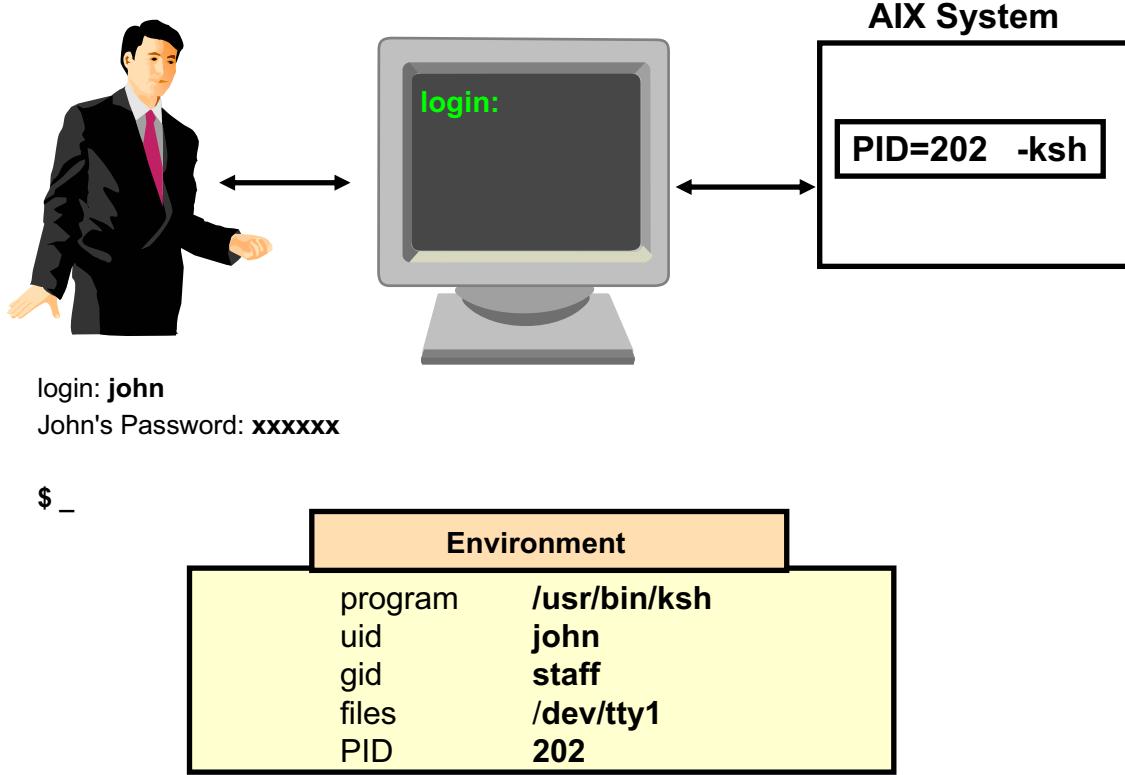
An active process has information held about it in the AIX kernel. The actual information will, of course, vary from one process to another, but in general the following will be held:

PROGRAM	name of the process
DATA	command line and options for the process
OPEN FILES	files being used by the process
CURR. DIR.	the current directory, from which the process was invoked
UID	the ID of the user that invoked the process
GID	the ID of the group that this user belongs to
PID	a unique number for this process at this point in time
PPID	the PID of the parent process that was/is responsible for this process
VARIABLES	any variable information

Additional Information —

Transition Statement — Having defined what a process is, let us take a closer look at the process environment.

Login Process Environment



© Copyright IBM Corporation 2008

Figure 10-3. Login Process Environment

AU1310.0

Notes:

Login shell

When you log in to a system, AIX starts a new process (in the example with PID=202) and loads the program `/usr/bin/ksh` into this process. This shell is called the *login shell*.

The PID is randomly allocated by the kernel.

Instructor Notes:

Purpose — Describe the login environment.

Details — The example on the visual describes the environment for the user's login shell. This environment contains all the information necessary for the process to run.

Whenever a user logs in to AIX, a Korn shell (**ksh**) is run for that user.

This shell's environment consists of the user name (**UID**), the group the user belongs to (**GID**), as well as the terminal that the user is logged on to (**FILES**).

This information will be unique to this shell as long as the user remains logged in.

Once the user logs out, this information will be forgotten until the user logs in again, at which point this information (the environment) will be recreated once more.

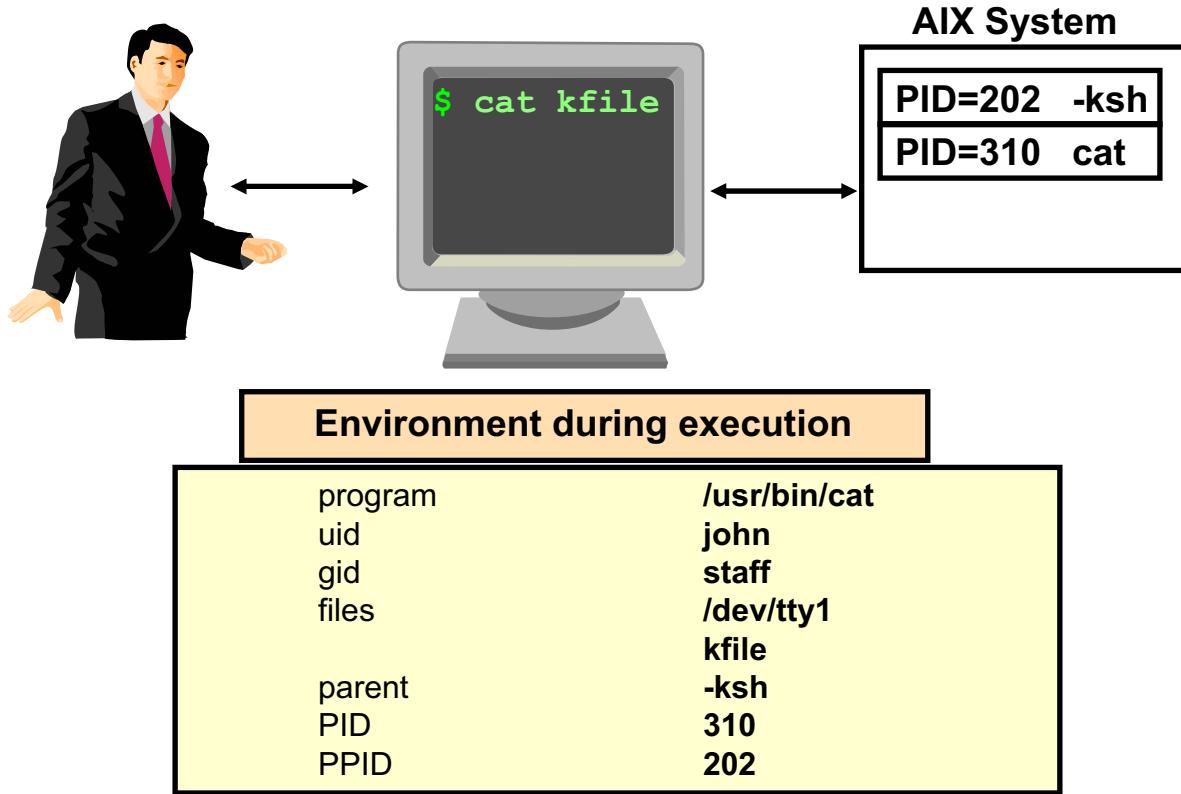
Discussion Items — In the visual what does the **/dev/tty1** refer to? The terminal that the user is logged in to.

One point about the question is that this file is then used by the shell to communicate with the user.

Additional Information —

Transition Statement — Each process has its own unique environment. This is largely defined for it by the process that invoked this process.

Process Environment



© Copyright IBM Corporation 2008

Figure 10-4. Process Environment

AU1310.0

Notes:

Process relationships

Processes exist in parent/child hierarchies. A process which starts or executes a program or a command is a parent process; a child process is the product of the parent process. A parent process may have several child processes, but a child process can only have one parent.

This process manifests itself when the user starts running commands after they are logged in to the system. The shell waits for instructions and having received them, executes them. The instructions usually involve starting up a process, like an editor. In this situation, the shell is the parent process and the editor becomes the child.

All child processes inherit an environment from their parent. This environment tells the child who invoked it, where to send output, and so forth.

Example

In the example, the user executes the command `cat kfile`. The shell uses the `PATH` variable to find the program `cat`. This program resides in directory `/usr/bin`. Afterwards, the shell starts a new process (`PID=310`) and loads the program `/usr/bin/cat` into this new process.

Instructor Notes:

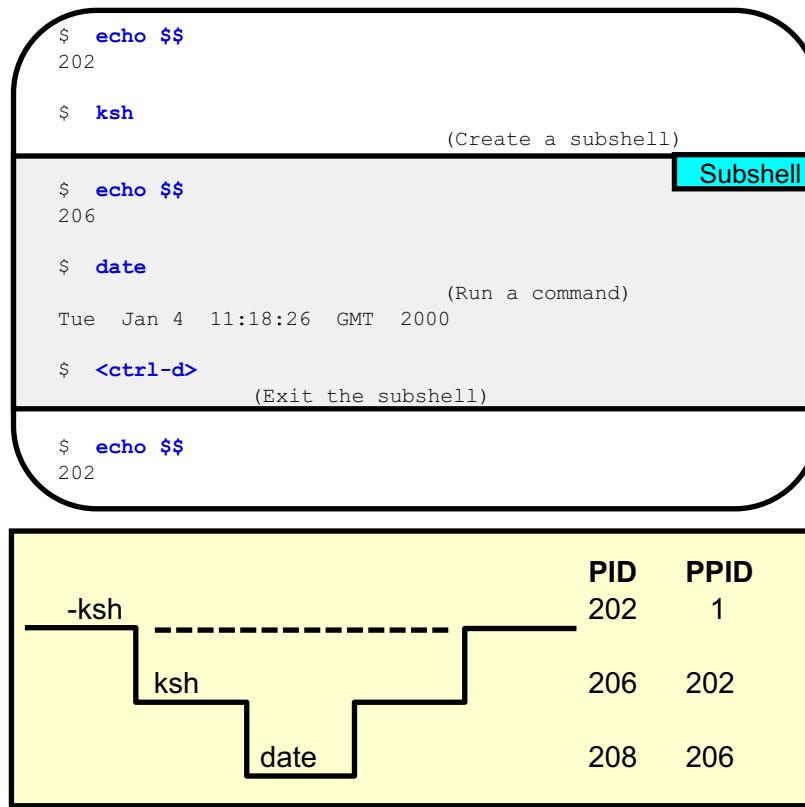
Purpose — Explain the environment of an executing program.

Details — It is important to point out that the environment that a child process receives is local to it. The child can modify that environment and pass it down to its children, but the modified environment cannot be passed back to the invoking parent, unless the child was invoked in the current shell through the use of “.” as described later.

Additional Information —

Transition Statement — Let us see an example of parent and child processes.

Parents and Children



© Copyright IBM Corporation 2008

Figure 10-5. Parents and Children

AU1310.0

Notes:

Parent versus child process

The **PID** is the process identification number used by the kernel to distinguish the different processes. PID 1 is always the **init** process which is the first AIX process that is started during the boot process.

The **PPID** is the parent process identification number, or in other words the PID of the process which started this one.

The special environment variable **\$\$** is mostly used within shell scripts to distinguish between multiple instances of the same shell script (for instance when unique temporary file names need to be used).

There are some exceptions. The **echo** command is built into the shell, so it does not need to create a subshell in which to run **echo**.

Examples

In the example above, a second **ksh** was started as a way to illustrate the parent/child relationship with processes. As another example, a second different shell could be started (for example, the **csh**) to run specific shell scripts or programs.

Instructor Notes:

Purpose — Demonstrate parent and child processes.

Details — This diagram illustrates what happens to the parent while the child is running.

In the example, we start from a login shell. It has a PID of 202.

Then a subshell is started with `ksh`. The PID is now 206.

From within the subshell another process is run.

While the `date` command runs, we cannot interact with the shell. Once the command has finished, we can run other processes, but not before. In effect, the parent sleeps while the child runs.

When the child process terminates, the parent will reawaken.

Students may ask the purpose of starting a second shell. In our example, we are doing it to prove the parent/child process relationship. However, there may be occasions when a second different shell is started so that specific shell scripts or programs can be run (for example, the `csh`). Also, when a shell script is run, it often “spawns” a second shell in which the shell commands are executed.

The special variable `$$` changes as our current operating environment moves from one process to another.

A good way to reinforce the concepts here is to focus on what process is issuing the command prompt to STDOUT and which is reading our STDIN.

Additional Information —

Transition Statement — Child processes are started by a parent. We shall now take a look at what happens to the process environment.

Variables and Processes

- **Variables** are part of the process **environment**.
- Processes **cannot access or change variables** from another process.

```

$ x=4
$ ksh
$ echo $x
Subshell
$ x=1
$ <ctrl-d>
$ echo $x
4

```

© Copyright IBM Corporation 2008

Figure 10-6. Variables and Processes

AU1310.0

Notes:

Process environment

Variables are local to the shell or process from which they are set. Child processes will not automatically inherit the variables of the parent. The variable `x` is not known in the subshell that has been started.

To pass variables into a subshell the `export` command must be executed. That is shown in the next activity.

Instructor Notes:

Purpose — Explain the relationship of variables in parent and child processes.

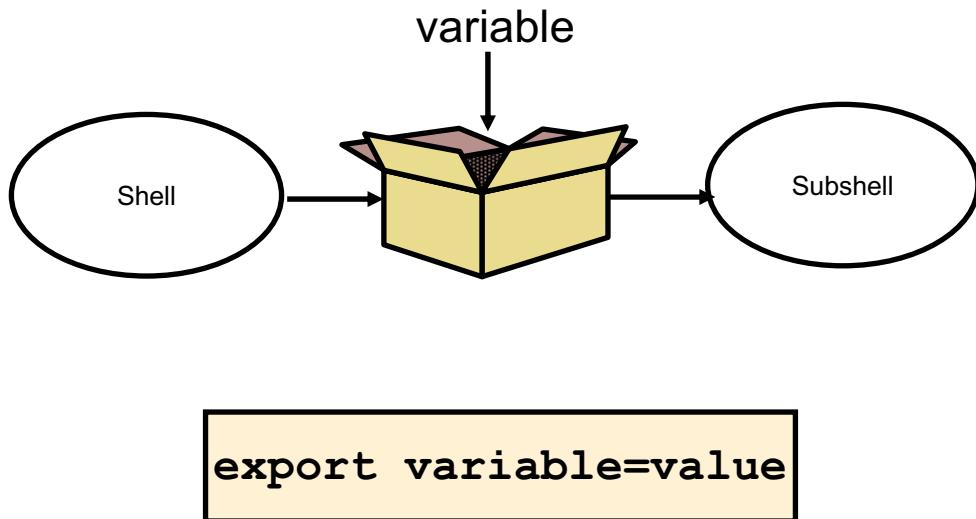
Details —

The example shows the problems with passing variables. In our login shell, we create a variable called `x` and give it the value of `4`. In a child shell, the variable does not exist. So, we can set it once more, but this time we give it a value of `1`. Returning to the parent shell, `x` is once again set to the original value of `4`.

Additional Information —

Transition Statement — As we have seen, variable settings are local to the shell in which they are created. We will now see how variables can be made available for any child process.

Activity: Exporting Variables



© Copyright IBM Corporation 2008

Figure 10-7. Activity: Exporting Variables

AU1310.0

Notes:

Activity

This activity introduces the **export** command.

- ___ 1. Log in to the system.
- ___ 2. Write down the process ID of your current shell.

Process ID:

- ___ 3. Define two shell variables `vartest1` and `vartest2` in the following way:

```
$ vartest1="moon"  
$ vartest2="mars"
```

Execute the **export** command only for variable `vartest2`.

- ___ 4. Print the value of `vartest1` and `vartest2`.

-
- ___ 5. Start a new shell:
 - ___ 6. Write down the process ID of the subshell.

Process ID:

- ___ 7. Check if the variables `vartest1` and `vartest2` are defined in your subshell.

- ___ 8. In your subshell change the value of variable `vartest2`:
- ___ 9. Exit your subshell and print out the value of `vartest2`.

Has the variable been changed in the parent shell?

- ___ 10. Please answer the following question to summarize this activity:

To pass variables into a subshell, which command must be executed?

Activity with Hints

This activity introduces the **export** command.

- ___ 1. Log in to the system.

 » login: **teamxx** (at the login prompt)
 Password: **teamxx** (default password same as user name)

- ___ 2. Write down the process ID of your current shell.

Process ID:

 » \$ **echo \$\$**

- ___ 3. Define two shell variables **vartest1** and **vartest2** in the following way:

 \$ **vartest1="moon"**
 \$ **vartest2="mars"**

Execute the **export** command only for variable **vartest2**.

 » \$ **export vartest2**

- ___ 4. Print the value of **vartest1** and **vartest2**.

 » \$ **echo \$vartest1**
 » \$ **echo \$vartest2**

- ___ 5. Start a new shell:

 » \$ **ksh**

- ___ 6. Write down the process ID of the subshell.

Process ID:

 » \$ **echo \$\$**

- ___ 7. Check if the variables **vartest1** and **vartest2** are defined in your subshell.

 » \$ **echo \$vartest1**
 » \$ **echo \$vartest2**

- ___ 8. In your subshell change the value of variable **vartest2**:

 » \$ **vartest2="jupiter"**

___ 9. Exit your subshell and print out the value of `vartest2`.

» \$ `exit`
» \$ `echo $vartest2`

Has the variable been changed in the parent shell?

» No, the variable has not been changed.

___ 10. Please answer the following question to summarize this activity:

To pass variables into a subshell, which command must be executed?

» You must use the `export` command.

Instructor Notes:

Purpose — Activity that introduces the `export` command.

Details — Give the students some time to work on this activity. Review the activity afterwards.

Additional Information — Sometimes we see this syntax:

```
$ var=<value> command <parameters>
```

This is the equivalent of doing:

```
$ ksh
$ export var=value
$ command <parameters>
$ exit
```

The main point is that the assigned value is only in effect during the execution of the command.

Transition Statement — Users can create programs called shell scripts that start up processes just as we have seen with system commands. Let's look closer at how this works.

What Is a Shell Script?

A **shell script** is a **collection of commands** stored in a text file

```
$ vi hello
```

```
echo "Hello, John. Today is: $(date)"  
pwd  
ls  
~  
~  
~  
:wq
```

```
$
```

© Copyright IBM Corporation 2008

Figure 10-8. What Is a Shell Script?

AU1310.0

Notes:

Creating a shell script

A shell script is a simple text file that contains AIX commands.

When a shell script is executed, the shell reads the file one line at a time and processes the commands in sequence.

Any AIX command can be run from within a shell script. There are also a number of built-in shell facilities which allow more complicated functions to be performed. These will be illustrated later.

Any AIX editor can be used to create a shell script.

Additional information

More information on Korn shell features such as aliasing can be found in the AIX 6.1 online documentation using search arguments such as Korn shell, **ksh**, and programming.

Instructor Notes:

Purpose — Define what a script is and how it works. This will then enable you to create scripts and to define how to invoke them.

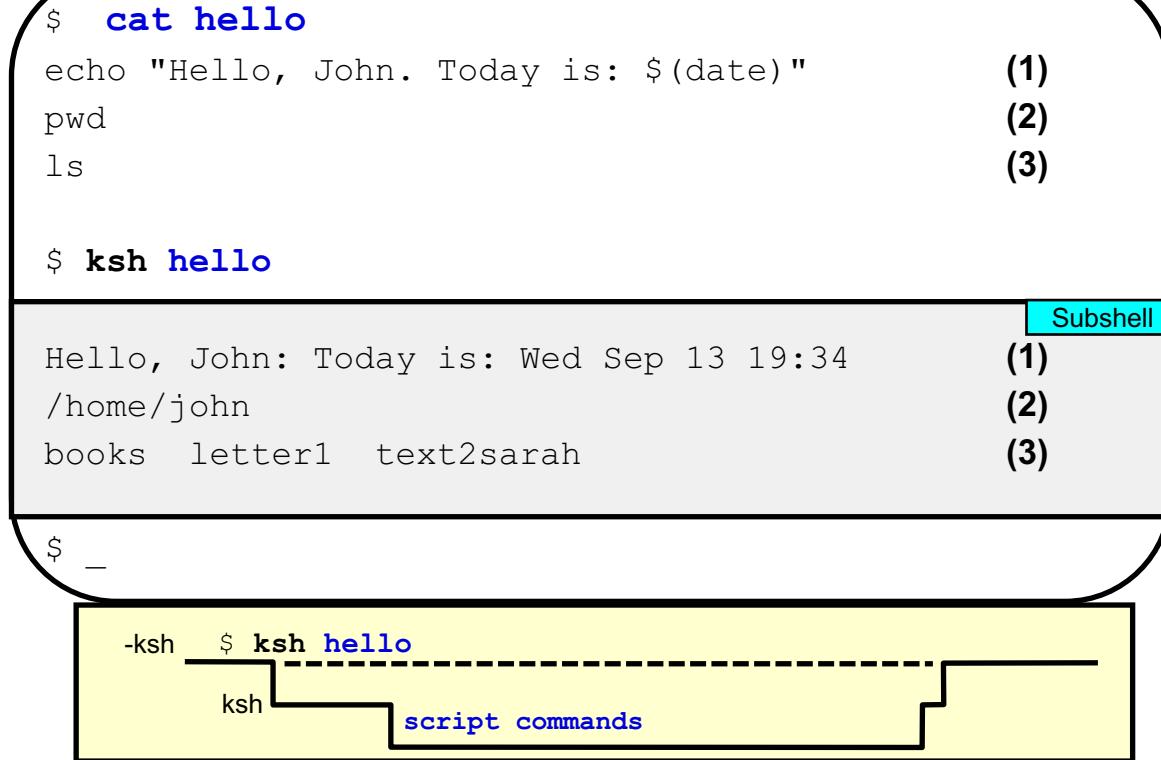
Details — The example in the visual shows three commands placed into a shell script.

Additional Information — Anything that can be done in the shell can also be done in a shell script, except managing an interactive command which prompts you for more input.

There are features of the shell that can be used in a script that cannot be accessed at the shell prompt.

Transition Statement — Now we know how to create a shell script, let's see how this system handles its execution.

Invoking Shell Scripts (1 of 3)



© Copyright IBM Corporation 2008

Figure 10-9. Invoking Shell Scripts (1 of 3)

AU1310.0

Notes:

Shell script example

A shell script is a collection of commands in a file. In the example a shell script **hello** is shown.

To execute this script, start the program **ksh** and pass the name of the shell script as argument:

```
$ ksh hello
```

This shell reads the commands from the script and executes all commands line by line.

Instructor Notes:

Purpose — Describe executing a shell script using the **ksh** command.

Details — Creating shell scripts is only half of the task. We also need to be able to invoke these scripts on demand. There are several ways of doing this.

One way of invoking a script is to invoke a shell that reads and interprets the script:

ksh script1

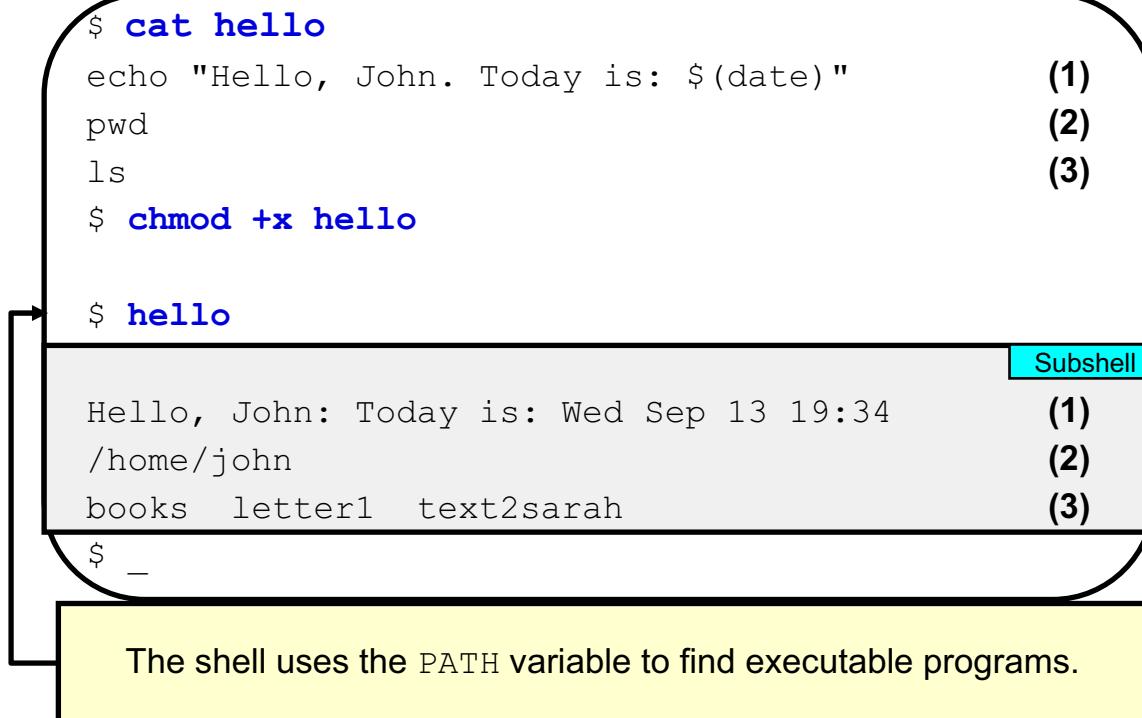
As described in the visual, this invokes a subshell and runs each command in the script in turn.

When the shell reaches the end of file in the script, the subshell will also terminate.

Additional Information —

Transition Statement — There are other ways of invoking scripts.

Invoking Shell Scripts (2 of 3)



© Copyright IBM Corporation 2008

Figure 10-10. Invoking Shell Scripts (2 of 3)

AU1310.0

Notes:

Executing shell scripts

This visual shows another way of invoking a shell script. This method relies on the user first making the script an executable file with the `chmod` command. After this step, the script can be invoked by its name.

Note that the shell uses the `PATH` variable to find executable files. If you get an error message like the following,

```
$ hello
ksh: hello: not found
```

check your `PATH` variable. The directory in which the shell script is stored must be defined in the `PATH` variable.

Instructor Notes:

Purpose — This visual demonstrates another way of invoking a script.

Details — Have a look to the previous page. The graphic to display the process hierarchy is the same. This method relies on the user first making the script an executable file with the `chmod` command.

`chmod` can be used with the symbolic format:

`chmod ugo+rwx script`

or with the octal format:

`chmod 755 script`

Once the first step has been carried out, then all that remains is to invoke it as if it were a normal command.

With the execute permission on the script, the current shell will automatically execute a subshell with the script name as the input argument. The result is the same as if we had run: `$ ksh scriptname`.

Discussion Item — What permissions are set by the command `chmod +x script1?`

Answer: Execute permissions are given to user, group, and other.

Additional Information — The users of the script *must* have both the READ as well as the EXECUTE permissions set. This is because the shell needs to open the script file in order to read the lines it contains.

Note that if the script is in a directory to which there is no path (see the `PATH` variable for a list of directories which will be searched by the shell for programs) then the command will fail with a message:

`ksh: <script>: not found`

Transition Statement — Both methods of running a script involve executing the script in a subshell. There is yet another way of running a script.

Invoking Shell Scripts (3 of 3)

```
$ cat set_dir
dir1=/tmp
dir2=/usr
```

```
$ . set_dir
$ echo $dir1
/tmp
$ echo $dir2
/usr
```

. (dot): Execution in the current shell



What is the value of `dir1` and `dir2`, if `set_dir` is called without the dot?

© Copyright IBM Corporation 2008

Figure 10-11. Invoking Shell Scripts (3 of 3)

AU1310.0

Notes:

Variables and shell scripts

Each shell script is executed in a subshell. Variables defined in a shell script cannot be passed back to the parent shell.

If you invoke a shell script with a . (dot), it runs in the current shell. Variables defined in this script (`dir1`, `dir2`) are therefore defined in the current shell.

Instructor Notes:

Purpose — Describe the final method of invoking a script in the current shell.

Details — No graphic to display the process hierarchy is shown. All commands of the script are executed within the shell.

The previous forms of running a shell script involved creating a subshell in which to run the commands. Any environment settings or changes are completely local to the script and will be forgotten when the script ends.

If changes need to be retained in the parent shell, then the script should be invoked by preceding it with a dot (.). This ensures that it runs in the current shell.

For example, this can be useful if changes have been made to a .profile. By typing:

```
. .profile
```

the shell will reexecute the profile for this user without having to log out.

It is a useful exercise to ask the students the meaning and significance of each of the three dots in the command line:

```
$ . ./profile
```

There are dangers involved in using this method by default: once settings are made in the current shell (usually the login shell), then the only way of returning to the default settings would be to log out and log in again.

Discussion Items — Before telling the student the usefulness of this method ask the students where they think this method could be used?

Additional Information —

Transition Statement — Processes inform the parent of how they ran, for example: did it succeed or fail?

We shall now look at these exit codes from processes and what can be done with them.

Exit Codes from Commands

- A command returns an **exit value** to the parent process:

0	=	Success
1 - 255	=	Other than successful

- The environment variable **\$?** contains the exit value of the last command:

```
$ cd /etc/security
ksh: /etc/security: Permission denied
$ echo $?
1
```

© Copyright IBM Corporation 2008

Figure 10-12. Exit Codes from Commands

AU1310.0

Notes:

Process exit codes

When a command executes, if it completes successfully, it returns to its parent shell the value of zero (0) which is stored in a variable **\$?**. This value is referred to as the return code or the exit code. If, however, the command completes unsuccessfully, a positive number between the range of 1 to 255 is returned.

To obtain the return code use the following: **\$ echo \$?**

```
$ date
$ echo $?
0
```

This shows successful execution of the **date** command. The visual shows an example for an unsuccessful execution of a command.

Instructor Notes:

Purpose — Introduce the concept of return codes.

Details —

Additional Information —

Transition Statement — Let's do a few checkpoint questions.

Checkpoint

1. When would you execute a shell script using the dot (.) notation? Why?
2. What is the command that is used to carry down the value of a variable into the subshell?
3. What would be the value of x at the end of the following steps?

```
$ ( ... login shell ... )  
$ ksh  
$ x=50  
$ export x  
$ <ctrl -d>  
$ (what is the value of x set to now?)
```

© Copyright IBM Corporation 2008

Figure 10-13. Checkpoint

AU1310.0

Notes:

Instructor Notes:

Purpose — To test the students' understanding of this unit's material.

Details —

Checkpoint Solutions

1. When would you execute a shell script using the dot (.) notation? Why? **When you are using the script to change variable values in the current shell.**
2. What is the command that is used to carry down the value of a variable into the subshell? **export variable_name**
3. What would be the value of x at the end of the following steps?

```
$ ( ... login shell ... )  
$ ksh  
$ x=50  
$ export x  
$ <ctrl -d>  
$ (what is the value of x set to now?)
```

x would have the value it had before starting the subshell. If the login shell had not set the variable, then after return from the subshell it would still not be set.

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — Before we wrap up this unit, let's do a short activity on shell scripts.

Activity: Shell Scripts



© Copyright IBM Corporation 2008

Figure 10-14. Activity: Shell Scripts

AU1310.0

Notes:

Activity

- 1. Log in to the system.
- 2. Create a shell script `count_files` that prints the number of files in the current directory.
- 3. Make the script executable.
- 4. Invoke the script. If the shell cannot find your script, check the `PATH` variable, or provide a path to the script on the command line.
- 5. Create another shell script called `active` that counts the number of active users.
- 6. Make the script executable and invoke it afterwards.

Activity with Hints

This activity introduces the **export** command.

- ___ 1. Log in to the system.

```
» login: teamxx (at the login prompt)  
Password: teamxx (default password same as user name)
```

- ___ 2. Create a shell script **count_files** that prints the number of files in the current directory.

```
» $ vi count_files  
echo "Number of files:"  
ls -l | wc -l
```

- ___ 3. Make the script executable.

```
» $ chmod u+x count_files
```

- ___ 4. Invoke the script. If the shell cannot find your script, check the **PATH** variable, or provide a path to the script on the command line.

```
» $ count_files
```

- ___ 5. Create another shell script called **active** that counts the number of active users.

```
» $ vi active  
echo "Active users:"  
who  
echo "Number of active users:"  
who | wc -l
```

- ___ 6. Make the script executable and invoke it afterwards.

```
» $ chmod u+x active  
» $ active
```

Instructor Notes:

Purpose — Activity to introduce shell scripts.

Details — Give the students some time to work on this activity. Review the scripts afterwards.

Additional Information —

Transition Statement — Before moving on to a new unit, let's summarize this unit's key points.

Unit Summary

- Shell scripts can be invoked in three ways:
 - \$ ksh scriptname (must have read permission)
 - \$ scriptname (must have read and execute permission)
 - \$. scriptname (must have read permission)
- Each program runs in an AIX process.
- Every process has an environment in which it runs much of which is inherited from its initiating process, the parent process.

© Copyright IBM Corporation 2008

Figure 10-15. Unit Summary

AU1310.0

Notes:

Instructor Notes:

Purpose — To review the key points of this unit with the students.

Details —

Additional Information —

Transition Statement — Let's move on to a new unit.

Unit 11. Controlling Processes

Estimated Time

00:45

What This Unit Is About

This unit describes how processes can be monitored and controlled.

What You Should Be Able to Do

After completing this unit, students should be able to:

- Describe process monitoring
- Invoke background processes
- Terminate processes
- List useful signals
- Use the `nohup` command
- Control jobs in the Korn shell

How You Will Check Your Progress

Accountability:

- Checkpoint questions
- Exercise

Unit Objectives

After completing this unit, you should be able to:

- Describe process monitoring
- Invoke background processes
- Terminate processes
- List useful signals
- Use the **nohup** command
- Control jobs in the Korn shell

© Copyright IBM Corporation 2008

Figure 11-1. Unit Objectives

AU1310.0

Notes:

Instructor Notes:

Purpose — To introduce this unit's objectives to the students.

Details —

Additional Information —

Transition Statement — Before we can do anything with a process, we first must be able to see what processes are active on the system. This can be done with the `ps` command.

Monitoring Processes

The **ps** command displays process status information.

\$ **ps -f**

UID	PID	PPID	...	TTY	...	COMMAND
john	202	1	...	tty0	...	-ksh
john	206	202	...	tty0	...	ksh
john	210	206	...	tty0	...	ls -R /
john	212	206	...	tty0	...	ps -f

© Copyright IBM Corporation 2008

Figure 11-2. Monitoring Processes

AU1310.0

Notes:

Displaying process status information

The **ps** command lists processes in the same manner as **ls** lists files. By default, it prints information only about processes started from your current terminal. Only the Process ID, Terminal, Elapsed Time and Command with options and arguments are displayed.

The **-e** option displays information about EVERY process running in the system.

The **-f** option in addition to the default information provided by **ps**, displays the User Name, PPID, start time for each process (that is, a FULL listing).

The **-l** option displays the user ID, PPID and priorities for each process in addition to the information provided by **ps** (that is, a LONG listing). It provides only the process name instead of the original command line.

In addition to these options, AIX has support for all of the System V options.

Instructor Notes:

Purpose — Discuss how to display process status.

Details — AIX allows a user to run, by default, up to 40 processes simultaneously. This means that the user may have to check how these are currently performing.

Processes run attached to a user (UID) or to a terminal (tty). The `ps` command allows the listing of processes based upon these, and other criteria.

There is a new option with AIX 5L V5.3 (`-P`) that displays the project name and subproject identifier for the project. This is probably too much information for the students at this point.

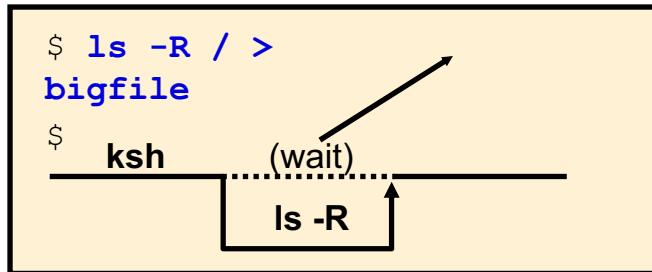
Another new option in AIX 5L V5.3 is the `-T` flag. This flag will produce a pseudo-graphic hierarchy listing of the targeted process and all of its children, recursively. Fun to run, and also useful if you need to understand the parent/child relationships of the processes on your system.

Additional Information — AIX 5L V5.2 added a System V `ps` command in `/usr/sysv/bin` to support all the System V options. Also, mention that the traditional AIX `/usr/bin/ps` has long supported the BSD (Berkeley) set of options (the ones without the `-` in front of them).

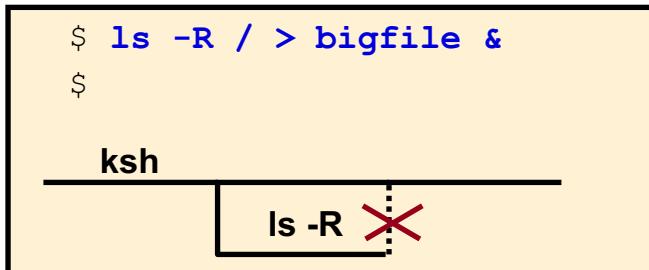
Transition Statement — Now that we can view the currently running processes, let's look at how they can be invoked to run in either the foreground or the background.

Controlling Processes

Foreground Processes:



Background Processes (&):



© Copyright IBM Corporation 2008

Figure 11-3. Controlling Processes

AU1310.0

Notes:

Starting processes

Processes can be invoked in different ways. If the command will finish in a short period of time, then we don't mind waiting for it to finish. On the other hand, if it is going to take minutes or hours to run, then we may wish to invoke it in such a way that we can continue to use the terminal.

Processes run in two states:

- Foreground: where they take full control over the terminal while they are running
- Background: where they run with no further interaction with the shell

Foreground processes

Processes that are started from and require interaction with the terminal are called *foreground processes*. Most important, the parent shell can not give you a new prompt until the foreground process completes.

Background processes

Processes that are run independently of the initiating terminal are referred to as *background processes*. A background process is launched as a sub-process, but the foreground (parent) shell immediately gets control back and issues a new prompt. This allows the user to continue to interact with the shell while the sub-process is executing. Background processes are most useful with commands that take a long time to run.

A process can only be run in the background if:

- i. It does not require keyboard input.
- ii. It is invoked with an ampersand (&) as the last character in the command line

Instructor Notes:

Purpose — Discuss foreground and background processes.

Details —

Additional Information — The last student note is not complete. The other way to place a process in the background is to suspend it and then use the `bg` command to place it in the background. It is not mentioned in the students note here because it would be premature; we will be covering it when we get to “job control” later in this unit.

Transition Statement — Running processes in the background could mean that the process is beyond the control of the user and if there was a problem, could run indefinitely.

It is important therefore, to be able to control both background and foreground processes.

Terminating Processes (1 of 2)

Foreground Processes:

<Ctrl-c> Interrupt key, cancels a foreground process

kill Sometimes the **kill** command is used to terminate foreground processes

Background Processes:

kill The **kill** command is the only way to terminate background processes

© Copyright IBM Corporation 2008

Figure 11-4. Terminating Processes (1 of 2)

AU1310.0

Notes:

Introduction

Sometimes a process, like **ls**, may take a long time to run. If you wish to stop a command before it completes, it can be stopped by breaking out of the command.

Stopping a foreground process

Foreground processes interact with the terminal. These can be stopped by a quit signal by pressing **<Ctrl-c>**. Sometimes, the **<Ctrl-c>** may not work. A shell script or program can trap the signal a **<Ctrl-c>** generates and ignore its meaning. In this case, you must use the **kill** command to terminate the process.

Stopping a background process

Background processes are not interacting with the terminal and must be stopped by using the **kill** command to terminate the process.

Instructor Notes:

Purpose — This visual explains how processes are stopped.

Details — If the foreground process does not respond to the <Ctrl-c>, stopping it may require logging in to another screen and running a command called `kill`. The `kill` command will be shown in more detail on the next visual.

Alternatively, if using a serial attached ASCII terminal, the user can always try switching off the screen and then switching it back on again. This would likely terminate your login shell and all of its children processes and give you a new login prompt.

When dealing with background tasks, the only sure way of stopping them (short of powering off the AIX system!) is to use the `kill` command.

Additional Information —

Transition Statement — Let us take a look at an example of how the `kill` command works.

Terminating Processes (2 of 2)

The **kill** command sends a signal to a running process, which normally stops the process.

```
$ ps -f
UID      PID  PPID   ...    TTY   ...    COMMAND
john    202      1   ...    tty0   ...    -ksh
john  204    202   ...    tty0   ...    db2_start
john  206    202   ...    tty0   ...    find /
```

\$ **kill 204** (Termination Signal)

\$ **kill -9 206** (Kill Signal)



Termination: Notification to the program to terminate
Kill: Kill the application **without notification**
(Use with care!)

© Copyright IBM Corporation 2008

Figure 11-5. Terminating Processes (2 of 2)

AU1310.0

Notes:

Introduction

The **kill** command is used to communicate a change of state to a command. The name of the **kill** command is misleading because many signals do not stop processes. The command can be used to tell the command or process to stop running but can also be used to convey other state changes to processes.

kill uses signals to communicate with the process. If no signal is specified, then the **kill** command issues a default signal, 15, to tell the process to terminate itself.

The example on the visual shows a **find** command running in the background. To end this process (as it may take a very long time to run), the command **kill** is used.

Who can stop processes?

A **root** user can stop any process with the **kill** command. If you are not a **root** user, you must have initiated the process in order for you to kill it.

Freeing up a hung terminal

If your terminal hangs, to clear the problem, try the interrupt key, <**ctrl-c**>, or try using <**Ctrl-q**> (in case the terminal output is suspended), or try using <**ctrl-d**> (in case it is just a foreground program waiting for more STDIN input).

If these actions still do not free the terminal, you can usually free up the terminal by logging in at a different terminal and using the **kill** command to kill the login shell of the hung terminal.

kill -9

A *kill signal* (-9) kills an application “with extreme prejudice.” This means that rather than the process terminating itself, the operating system terminates the process. When a process is killed in this manner, it does not have an opportunity to close any open files, or save any data, and can lead to corruption. For example, if you kill a database server process, you might end up with a corrupt database. Always try to stop processes by sending a normal *termination signal* (no flag, or -15), and use the (-9) signal as a last resort.

Instructor Notes:

Purpose — Here we shall see how to use the `kill` command to stop a background process.

Details —

Additional Information — There used to be a possibility of creating zombie processes on the system by use of the `kill -9` command. This has sometimes been known to occur with “home grown” or purchased application packages. These defunct processes are deleted from the system only after a system reboot or occasionally by cleanup processes initiated by the `init` process. This problem seems much better with later releases of AIX 5L.

Transition Statement — `kill` has a large number of signals that may be sent to it. We shall now look at some of these.

Signals

Signal	Meaning
1	hangup - you logged out while the process was still running
2	interrupt - you pressed the interrupt (break) key sequence <Ctrl+c>
3	quit - you pressed the quit key sequence <Ctrl+\>
9	Kill signal: The most powerful (and risky) signal that can be sent: Signal cannot be avoided or ignored!
15	Termination signal (Default): Stop a process Signal can be handled by programs

© Copyright IBM Corporation 2008

Figure 11-6. Signals

AU1310.0

Notes:

Introduction

Signals are used to communicate a change of state to a command. This may mean that the command or process should stop running or it may even mean that this process should re-read its parameter files.

Signals

Signals are used to tell the **kill** command what to do with the PID specified in the command. By default, the **kill** command sends a signal of 15 to a process.

To send a different signal to a process use **kill -num PID** where *num* is the signal that you want to send.

The HANGUP signal (1) is sent to a process if its parent dies, for example if you log off when a background process is running.

The INTerrupt signal (2) is generated when the user presses the interrupt key (**Ctrl-c**) on the keyboard.

The QUIT signal (3) is generated by the user pressing the quit key <**ctrl-\>**.

The most powerful signal you can send to a process is the KILL signal (9), which is sent to all processes when the system is shutting down. Processes which refuse to be killed by other signals will usually be killed by **kill -9 PID**.

Listing signals

To list all the signals supported use the **kill -l** command. From this list you can also specify the **kill** command with the name of the signal rather than the number. For example, signal 3 refers to the Quit signal, so you could enter **\$ kill -QUIT** rather than **\$ kill -3**.

Note that the number of the signal bears no resemblance to its strength or priority.

Instructor Notes:

Purpose — The `kill` command works with signals. This visual describes these signals.

Details — If the command `kill PID` did not work, then the user has to use a signal to try and force the process to terminate.

Useful signals are:

<code>kill -1 PID</code>	Hangup, or what happens when the user logs off with a running background process
<code>kill -2 PID</code>	Same as pressing <ctrl-c>
<code>kill -3 PID</code>	Same as pressing <ctrl-\>, but if successful, it will create a core file
<code>kill -9 PID</code>	Die! Processes cannot ignore this signal.
<code>kill -15 PID</code>	The default. Same as <code>kill PID</code> .

Additional Information — Officially, the correct syntax for the `kill` command is:

`kill -s <signal> pid`

The syntax without the `-s` flag is considered obsolete, though it is still supported and is still the more commonly used syntax.

Transition Statement — Sometimes we may wish to keep processes running even after we log off. We shall now see how this is done.

Running Long Processes

The **nohup** command will prevent a process from being killed if you log off the system before it completes:

```
$ nohup ls -R / > out 2> err.file &
[1]      59
$
```

If you do not redirect output, **nohup** will redirect output to a file **nohup.out**:

```
$ nohup ls -R / &
[1]      61
Sending output to nohup.out
$
```

© Copyright IBM Corporation 2008

Figure 11-7. Running Long Processes

AU1310.0

Notes:

Introduction

The **nohup**, or “no hangup” command, will take over control of a background process once the process has been invoked. It tells the process to ignore signals 1 and 3 (hangup and quit). This will allow the process to continue if you log off the system.

nohup is designed to be used for background processes as it has little meaning when used with a foreground process.

Command output

A process started by **nohup** cannot send its output to your terminal. If you do not redirect its output, **nohup** will redirect the output of the command to a file called **nohup.out**, located in the current working directory.

If more than one background process is started with **nohup** with the same current directory and the output has not been redirected, the **nohup.out** file will contain the

output from all those processes (either mixed or appended). For this reason, it is a good idea to redirect output when using `nohup`.

The output from a command may be redirected to a log file or even to the null device (`/dev/null`) if no output is required.

STDERR

If the standard error is a terminal, all output written by the named command to its standard error is redirected to the same file descriptor as the standard output.

Who owns the process after you log out?

Since all processes need to have a parent process associated with it, commands started with `nohup` will be connected to the `init` process as the parent when you log off the system.

Instructor Notes:

Purpose — This visual demonstrates how to run a process without having to remain logged in while it is running.

Details — All output from `nohup` commands goes to a file called `nohup.out` located in the directory in which the `nohup` command was invoked, unless it is redirected. It is recommended that output be redirected when using `nohup` to prevent unpredictable output results.

Additional Information — When running background commands, the user will always see a few numbers under the command:

[1] 567

these mean:

[1] this is the job number assigned to the background process

567 is the process ID (PID)

The job number is a not system wide unique process identifier (like the `pid`) but is only a unique identifier for each individual user to help simplify management of that user's jobs.

Discussion Items — What happens if several background processes are started with the `nohup` command and the output of each has not been redirected?

The answer is that the output would all be merged unpredictably in one file. This might get a bit confusing!

Transition Statement — We shall now see how to control background and foreground tasks in the Korn shell.

Job Control in the Korn Shell

<code>jobs</code>	Lists all jobs running in the background and stopped processes
<code><Ctrl-z></code>	Suspends foreground task
<code>fg %<jobnumber></code>	Executes job in foreground
<code>bg %jobnumber</code>	Executes job in background

© Copyright IBM Corporation 2008

Figure 11-8. Job Control in the Korn Shell

AU1310.0

Notes:

Finding background processes

When running multiple processes, the trick is to identify which processes are running in the background. By using the `ps` command, it is not normally possible to identify these background processes. There is a tool specifically designed for locating these processes called `jobs`. This example shows that two processes are running in the background.

```
$ jobs
[2] +Running ls -R / > outfile &
[1] - Running ls -R / > outfile1 &
```

Stopping background processes

The number between the brackets is used when controlling the background process by referring to it by %jobno, for example, **kill %1** would stop the process labeled as job number one.

Moving a foreground process to the background

You can suspend a foreground process by pressing <Ctrl-z>. The running process is paused, and the parent shell is re-activated. The kernel is maintaining the environment of the process and able to resume the execution at the point where it was suspended.

To resume a suspended processes in the background, use the **bg** command. To bring a suspended or background process into the foreground, use the **fg** command.

The **bg**, **fg**, and **kill** commands can be used with a job number. For instance, to bring job number 3 from the background into the foreground, you can issue the command:

```
$ fg %3
```

nohup command

The **jobs** command does not list jobs that were started with the **nohup** command if the user has logged off and then logged back in to the system. On the other hand, if a user invokes a job with the **nohup** command and then issues the **jobs** command without logging off, the job will be listed.

If you started a job that is taking longer than you expected and need to log off, you can apply the **nohup** command to an existing process. For example, the user started a job in the background but forgot to add the **nohup** command:

```
$ start_app >/home/team01/app.out &
```

If the user logs off, the application will terminate. To allow it to continue to execute, add the **nohup** command to that process by finding its PID and then issuing the **nohup** command:

```
$ jobs
$ ps
  PID   TTY  TIME CMD
 5314  pts/0  0:00 -ksh
 11522 pts/0  0:00 ps
 19208 pts/0  0:01 start_app
$ nohup -p 19208
```

The user can now log off and the job will continue to execute until completion.

Instructor Notes:

Purpose — Explain how foreground and background tasks are controlled.

Details — The visual refers to using a control sequence <Ctrl-z>, which is used to suspend the currently running foreground task.

Job control is the facility which supports the use of the **jobs**, **bg**, and **fg** commands along with the use of <Ctrl-z> and the ability to reference jobs numbers instead of process IDs.

Job control is turned on by default. To turn job control off:

```
set +o monitor (plus sign) .
```

To turn back on:

```
set -o monitor
```

On some older ASCII screens, the <Ctrl-z> key sequence may not suspend a foreground job. If this is the case, enter the following:

```
stty susp <Ctrl-z>
```

The purpose of this is to force the shell to use the key value stored as <Ctrl-z>.

Additional Information —

Transition Statement — We shall now take a look at an example of job control.

Job Control Example

```
$ ls -R / > out 2> errfile & Start job
[1]      273

$ jobs      Lists jobs
[1] +      Running      ls -R / > out 2> errfile &
$

$ fg %1      Foreground
ls -R / > out 2> errfile

<ctrl-z>    Suspend
[1] +      Stopped (SIGTSTP) ls -R / > out 2> errfile &
$

$ bg %1      Background
$ jobs      Lists jobs
[1] +      Running      ls -R / > out 2> errfile &
$

$ kill %1    Terminate
[1] +      Terminate      ls -R / > out 2> errfile &
$
```

© Copyright IBM Corporation 2008

Figure 11-9. Job Control Example

AU1310.0

Notes:

Introduction

This visual shows how you can work with job control commands in a Korn shell.

Instructor Notes:

Purpose — This visual shows how job control can be achieved with the tools described.

Details — Depending on the level of your audience, introduce the `sleep` command in addition to the `ls` command.

`sleep` produces no output and no error messages, so it is ideal for this example.

Use `sleep` with a large number, such as 2000. This will suspend the screen process for 2000 seconds unless the job is put into background.

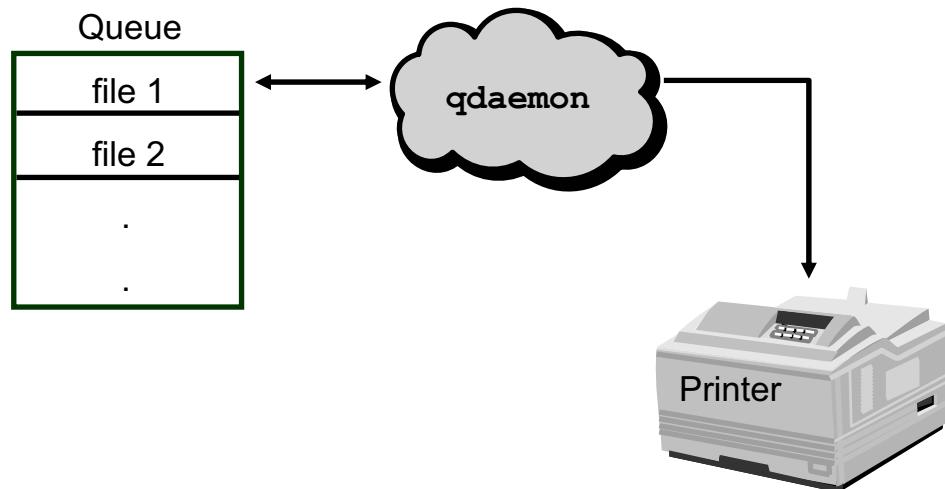
Additional Information — As per the visual, the `ls` command will produce lots of error messages. This might be a good time to remind the students about redirecting STDERR.

Transition Statement — Background processes fall into two categories, one which we have already seen: user tasks.

The other category refers to processes that were not started by a user.

Daemons

A **daemon** is a never-ending process, that controls a system resource such as the printer queue.



© Copyright IBM Corporation 2008

Figure 11-10. Daemons

AU1310.0

Notes:

Introduction

A *daemon* (pronounced “day-mon”) is a process that usually starts when you start your system and runs until you shut it down. Daemons are processes that wait for an event to take place. Once an event is detected, then the daemon will take responsibility for the task and process it.

Daemon example

`qdaemon` is one example of a daemon. `qdaemon` tracks print job requests and the printers available to handle them. The `qdaemon` maintains queues of outstanding requests and sends them to the proper device at the proper time.

The common daemons are `cron`, `qdaemon`, and `errdemon`. There are others daemons as well, many of them providing network services.

Instructor Notes:

Purpose — This visual introduces the concept of processes that were not started by a user, that are not associated with a terminal, yet are running.

Details — Daemons are usually started at operating system startup time. There are configuration files that contain the daemons to be started, the tasks that these perform and so on. These files are maintained by the system administrator.

Additional Information — In the system administration classes we teach the students how to start and stop daemons using the System Resource Controller facility. There are circumstance when an administrator will start a daemon long after system boot, may choose not to have it start at system restart and may choose to stop the daemon. The point is that not all daemons start with the system boot and stay running all the time.

Transition Statement — Before we switch over to the lab, a few checkpoint questions.

Checkpoint

1. What option would you use with the `ps` command to show the detailed commands that you are running?
2. True or false? As an ordinary user, you can only kill your own jobs and not those of other users.
3. Which is the strongest signal that can be sent to a process to terminate it?
4. It is always sensible to start long jobs in the background with the `nohup` command. Why is this?
5. What is the name for special never-ending system processes in the UNIX environment?

© Copyright IBM Corporation 2008

Figure 11-11. Checkpoint

AU1310.0

Notes:

Instructor Notes:

Purpose — To test the students' knowledge of the presented material.

Details —

Checkpoint Solutions

1. What option would you use with the **ps** command to show the detailed commands that you are running? **ps -f**
2. True or false? As an ordinary user, you can only kill your own jobs and not those of other users. **True**
3. Which is the strongest signal that can be sent to a process to terminate it? **signal 9**
4. It is always sensible to start long jobs in the background with the **nohup** command. Why is this? **The job will not lock up the user's terminal, and will continue to run when you log off the system.**
5. What is the name for special never-ending system processes in the UNIX environment? **Daemons**

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — Time to test what you have learned in this unit with a lab.

Exercise: Controlling Processes



© Copyright IBM Corporation 2008

Figure 11-12. Exercise: Controlling Processes

AU1310.0

Notes:

After completing the lab, you will be able to:

- Monitor processes by using the `ps` or `jobs` command
- Control processes

Instructor Notes:

Purpose — Cover the goals of the lab.

Details —

Additional Information —

Transition Statement — Now, a summary of this unit's important points.

Unit Summary

- To monitor processes use the `ps` command.
- Background processes are invoked by including an ampersand & at the end of the command.
- Use the `kill` command to terminate processes.
- Some useful signals that terminate processes are `kill -2`, `kill -3`, and `kill -9`.
- Jobs can be controlled in the Korn shell by suspending a job with `<ctrl z>` and restarted using the `bg` or `fg` commands.
- The `nohup` command allows you to start a job in the background and complete processing after you log off.
- System processes are called `daemons` and are often used to control system resources like the printer queueing mechanism.

© Copyright IBM Corporation 2008

Figure 11-13. Unit Summary

AU1310.0

Notes:

Instructor Notes:

Purpose — To summarize this unit's key points.

Details —

Additional Information —

Transition Statement — Let's move on to a new unit.

Unit 12. Customizing the User Environment

Estimated Time

00:30

What This Unit Is About

This unit demonstrates how users' environments can be customized to meet their specific preferences.

What You Should Be Able to Do

After completing this unit, students should be able to:

- Describe the purpose of the login profile
- Change the `PATH` and `PS1` variables
- Use the shell history mechanism
- Set aliases for commonly used commands

How You Will Check Your Progress

- Checkpoint questions
- Exercise

Unit Objectives

After completing this unit, you should be able to:

- Describe the purpose of the login **profile**
- Change the **PATH** and **PS1** variables
- Use the shell **history** mechanism
- Set **aliases** for commonly used commands

© Copyright IBM Corporation 2008

Figure 12-1. Unit Objectives

AU1310.0

Notes:

Instructor Notes:

Purpose — To introduce the objectives that this unit will address.

Details —

Additional Information —

Transition Statement — When you log in to an AIX system, there are files that can control how your environment is set up.

Login Files

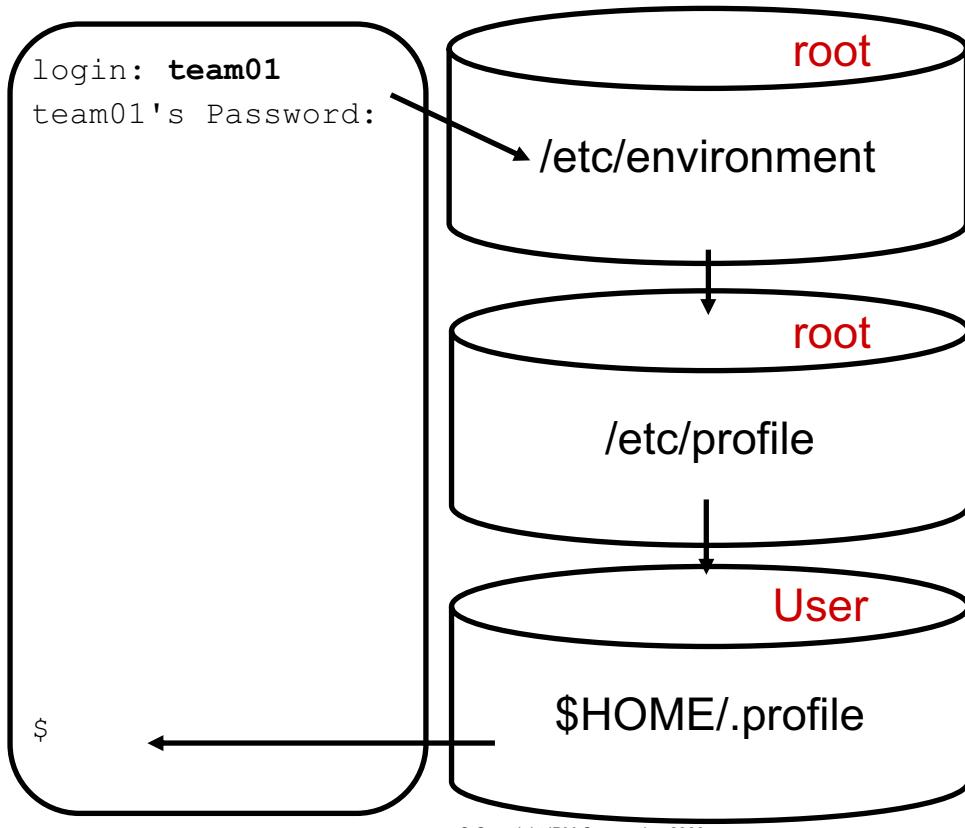


Figure 12-2. Login Files

AU1310.0

Notes:

Introduction

When you first log in to an AIX system, you have an opportunity to configure various settings that will control the way your shell session will work. Your environment is configured through several files that are read during the login process.

/etc/environment

The first file that the operating system uses at login is the **/etc/environment** file. This file contains variables specifying the basic environment for all processes and can only be changed by the system administrator.

/etc/profile

The second file that the operating system uses at login time is the **/etc/profile** file. This script controls system-wide default variables such as the mail messages and terminal types. It can only be changed by the administrator.

.profile

The **.profile** file is the third file read at login time. It resides in a user's login directory and enables a user to customize their individual working environment. The **.profile** file overrides commands run and variables set and exported by the **/etc/profile** file.

The contents of the **.profile** file can be any commands or settings that you would otherwise have to enter manually each time you log in to the system.

When setting variables in your **.profile** file, ensure that newly created variables do not conflict with standard variables such as MAIL, PS1, PS2, and so forth.

Instructor Notes:

Purpose — The visual outlines the files that are read as the user logs in. These files contain the environment that the user will get after logging on.

Details — The **.profile** file is normally not displayed with an **ls** command unless the **-a** option is given to **ls** as the file name starts with a dot (.)

Together, these profiles produce the user's environment.

Be sure students understand that the **/etc/environment** and **/etc/profile** files are changed only by the system administrator. The user, however, can change their own **.profile** file.

If students are familiar with CDE, you should mention that by default in CDE, the **.profile** file will not be read; the **.dtprofile** file will be used instead. More information on this is covered in the appendix.

Additional Information — Refer to Web-based documentation for an overview of security and access control for further details of the user login process.

Transition Statement — Let's take a look at a sample **/etc/environment** file.

Sample /etc/environment

\$ **cat /etc/environment**

```
# WARNING: This file is only for establishing environment
# variables. Execution of commands from this file or any
# lines other than specified above may cause failure of the
# initialization process.
```

```
PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:/sbin:
/usr/java131/jre/bin:/usr/java131/bin
TZ=EST5EDT
LANG=en_US
LOCPATH=/usr/lib/nls/loc
NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/%L/%N.cat
```

© Copyright IBM Corporation 2008

Figure 12-3. Sample /etc/environment

AU1310.0

Notes:

Establishing a standard environment

The **/etc/environment** file contains default variables set for each process. Only the system administrator can change this file.

/etc/environment variables

PATH is the sequence of directories that is searched when looking for a command whose path name is incomplete.

TZ is the time zone information.

LANG is the locale name currently in effect.

LOCPATH is the full path name of the location of National Language Support information, part of this being the National Language Support Table.

NLSPATH is the full path name for messages.

Instructor Notes:

Purpose — Explain the use of the **/etc/environment** file.

Details — The students may wish to display the file to view the entire contents.

The emphasis here is that these variables are set during user login. Do not spend a lot of time on this file as it cannot be altered by a normal user.

Point out that the lines beginning with the # are comments.

Transition Statement — Let's look at a sample **/etc/profile** file.

Sample /etc/profile

```
$ cat /etc/profile
.
.
#
# System-wide profile. All variables set here may be overridden by
# a user's personal .profile file in their $HOME directory. However
# all commands here will be executed at login regardless.
trap "" 1 2 3
readonly LOGNAME
# Automatic logout (after 120 seconds inactive)
TMOUT=120
# The MAILMSG will be printed by the shell every MAILCHECK seconds
# (default 600) if there is mail in the MAIL system mailbox.
MAIL=/usr/spool/mail/$LOGNAME
MAILMSG="[YOU HAVE NEW MAIL]"
# If termdef command returns terminal type (i.e. a non NULL value),
# set TERM to the returned value, else set TERM to default lft.
TERM_DEFAULT=lft
TERM=`termdef`
TERM=${TERM:-$TERM_DEFAULT}
.
.
export LOGNAME MAIL MAILMSG TERM TMOUT
trap 1 2 3
```

© Copyright IBM Corporation 2008

Figure 12-4. Sample /etc/profile

AU1310.0

Notes:

/etc/profile

The **/etc/profile** file contains the set of environment variables and commands that will be invoked when a user logs in to the system. These are settings that all users will have applied to their shell as they log in.

Any settings here can be overridden by a user's **.profile**.

Instructor Notes:

Purpose — The **/etc/profile** is a system-wide profile. It contains settings for all users of AIX. These settings may be overridden by a user's **.profile**.

Details — The settings contained here are normally environment variables but there may be one or two commands as well.

All variables must be exported in order to be made available to other shells.

Additional Information — The **readonly** command is a feature of the shell which allows the creation of read only variables. This may be useful where the administrator does not wish the user to customize their environment.

If the students ask about the **trap** commands, explain that it allows the script to intercept signals (passed with **kill**) and perform their own exit routines. In the example we are choosing to ignore the hang up, interrupt, and quit signals.

The **/etc/profile** file is owned by the **root** user and, therefore, cannot be modified by regular users.

Transition Statement — There are many variable settings in this file; let's take a look at these and see what they do.

Environment Variables (1 of 2)

LOGNAME	This holds your login name. It is read by many commands. Value cannot be changed (readonly variable).
TMOUT	Holds the value for how long a terminal can be inactive before the terminal is logged off by the system.
MAIL	Holds the name of the file where your mail is sent.
TERM	The terminal type you are using. Used by screen-oriented applications like vi or smit .

© Copyright IBM Corporation 2008

Figure 12-5. Environment Variables (1 of 2)

AU1310.0

Notes:

Typical /etc/profile file variables

These are some of the variables that can be found in the **/etc/profile** file:

- **MAIL** is the name of the file used by the mail system to detect the arrival of new mail.
- You can force a terminal to log off after a period of inactivity by setting the **TMOUT** variable in the **/etc/profile** file.
- The **MAILCHECK** variable specifies how often (in seconds) the shell will check for changes in the modification time of any of the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds.
- **MAILMSG** is the variable which holds the message you receive to tell you new mail has arrived.
- **LOGNAME** is the variable that the user logs in with.
- **TERM** is the variable that stores the terminal type.

Instructor Notes:

Purpose — To describe some of the variables found in the **/etc/profile** file.

Details — A typical **/etc/profile** will hold much many more settings and therefore will probably look rather different.

Note: The **/etc/profile** is owned by the root and therefore cannot be modified by normal users.

Additional Information —

Transition Statement — Let's look at a sample **.profile** file.

Sample .profile

```
$ cat .profile

PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:$HOME/bin:/usr/bin/X11:/sbin:.

PS1=' $PWD => '

ENV="$HOME/.kshrc" ← Execute this file every time a
                      new Korn shell is started.

export PATH PS1 ENV

if          [           -s "$MAIL" ]
then        echo      "$MAILMSG"

fi
```

© Copyright IBM Corporation 2008

Figure 12-6. Sample .profile

AU1310.0

Notes:

Controlling your shell with .profile

The **.profile** is a user-specific profile. It contains settings for individual users of AIX. The settings in this file will be acted on as the user logs in. These settings override any prior settings made in the **/etc/profile**. The **.profile** file is read only when the user logs in.

At startup time, the shell checks to see if there is any new mail in **/usr/spool/mail/\$LOGNAME**. If there is then **MAILMSG** is echoed back. In normal operation, the shell checks periodically.

The **ENV="\$HOME/.kshrc"** variable will cause the file **\$HOME/.kshrc** to be run every time a new Korn shell is explicitly started. This file will usually contain Korn shell commands.

Instructor Notes:

Purpose — To introduce the way a user can override the **/etc/profile** settings with the **.profile** file.

Details — A number of variables are set up for each user. The user may then decide to change these at any time. If users have further actions that need to be performed on their behalf, they can add these to their **.profile** files.

These will then be “remembered” at each login.

Caution should be taken when setting variables and shell properties. The **.profile** file is used by the bsh and ksh shells, yet the shells are not equal.

- Korn shell preferences should be set in **\$HOME/.kshrc**
- Bash preferences should be set in **\$HOME/.bshrc** and
- c-shell preferences should be set in **\$HOME/.cshrc**

Additional Information — Most, if not all, of these settings could have been made at the system-wide level.

If using CDE, by default, environment variables set in **.profile** are not used in CDE. To force CDE to also read the **.profile** file, uncomment the last line of the **.dtprofile** file so that **DTSOURCEPROFILE=true**.

Transition Statement — Let us take a look at what these variables and settings do.

Environment Variables (2 of 2)

PATH	A list of colon-separated directories that the shell searches for commands: PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:\$HOME/bin:/usr/bin/X11:/sbin:..
PS1	Primary system prompt (default= \$). To show the hostname and the current directory in the prompt: PS1="\$ (hostname) , '\$PWD: '
ENV	Pointer to a file containing Korn shell settings: ENV="\$HOME/.kshrc"

© Copyright IBM Corporation 2008

Figure 12-7. Environment Variables (2 of 2)

AU1310.0

Notes:

More environment variables

The PATH variable defines the search path for the directory containing commands and executable programs such as **ls**. Alternative directory names are separated with a : (colon). The current directory can be specified by two or more adjacent colons, or by a :: (colon period) as shown in the example above.

The current directory can also be specified by placing a . within two colons in the PATH variable:

/usr/bin:/etc:::/home/nick

PS1 is the shell prompt and is normally set to \$ for a user and # for root. It can be set to any string of text or to a variable.

PWD is a variable containing the current working directory.

MAIL is a pointer to the location of the user's mail directory.

MAILMSG is a string of text, normally set to You have new mail, that is displayed if the user has new mail in their mailbox.

ENV is a pointer to a file containing Korn shell settings. These cannot be exported like variables, so a variable is set up to reference a file containing these settings. In this way, each time a subshell is started, it will contain those settings automatically. This is covered in more detail in the next visual.

Instructor Notes:

Purpose — To consider a few of the environmental variables.

Details — Cover these additional environment variables.

Additional Information —

Transition Statement — Let us take a look at a Korn shell environment file.

Sample .kshrc

```
$ cat .kshrc

# set up the command recall facility
set -o vi

# set up a few aliases
alias ll='ls      -l'
alias p='ps -f'
alias up='cd      ..'
```

© Copyright IBM Corporation 2008

Figure 12-8. Sample .kshrc

AU1310.0

Notes:

Example of the .kshrc file

The `ENV` variable specifies a Korn shell script to be invoked every time a new shell is created. The shell script in this example is `.kshrc` (which is the standard name used), but any other filename can also be used.

The difference between `.profile` and `.kshrc` is that `.kshrc` is read each time a subshell is spawned, whereas `.profile` is read once at login.

You can also set the following variable in `$HOME/.profile`:

```
EDITOR=/usr/bin/vi
export EDITOR
```

It will do the same thing that the `set -o vi` command does as shown in the example.

Instructor Notes:

Purpose — The file pointed to be the ENV variable should contain Korn shell specific settings and commands.

Details — The problem with some of these settings is that they are not “remembered” across shell sessions.

The contents of this file are read every time a shell is started or a shell script is invoked thus solving that problem, unlike the **.profile** which is read only at login.

The name of the file is by convention **.kshrc**, but it can have any name.

The entries here are shell settings and aliases.

Discussion Items — Some students may ask what will happen if **set -o vi** is put in the **.profile** file instead of in **.kshrc**. This will allow command recall to work only in the login shell. Placing this command in the **.kshrc** file and then setting and exporting the **ENV** variable will allow command recall to be available in any subshell, for example, in any window if using AIXwindows, not just in the login shell.

Additional Information — For details on these and other special Korn shell settings, refer to the man pages or to the AIX online documentation.

Transition Statement — We shall now take a look at setting up aliases.

ksh Features - Aliases

```
$ alias p='ps -ef'
$ alias ll='ls -l'
```

```
$ alias ← Show all alias definitions
history='fc -l'
ll='ls -l'
p='ps -ef'
r='fc -e -'
```

© Copyright IBM Corporation 2008

Figure 12-9. ksh Features - Aliases

AU1310.0

Notes:

Introduction

Aliases are settings that contain complex commands that are commonly used. The alias name will normally be a mnemonic or a shorthand for the command that it symbolizes. The command or commands, are then assigned to the alias. From this point on, the alias contains the commands.

Assigning aliases

As shown in the visual, an alias can be set by simply typing the alias command followed by the mnemonic and a command or set of commands that are to be assigned to that mnemonic. The command or set of commands are in single quotes.

Predefined aliases

The `alias` command invoked with no arguments prints the list of aliases in the form name=value on standard output.

The Korn shell sets up a number of aliases by default. Notice that the `history` and `r` commands are in fact aliases of the `fc` command. Once this alias is established, typing an `r` will re-execute the previously entered command.

Passing aliases to subshells

Aliases can not be exported to subshells in the same manner as shell variables. To allow aliases to be set in subshells, the `ENV` variable has to be modified. The `ENV` variable is normally set to `$HOME/.kshrc` in the `.profile` file (although you can set `ENV` to any shell script). By adding the alias definition to the `.kshrc` file (by using one of the editors) and invoking the `.profile` file, the value of the alias will be carried down to all subshells, because the `.kshrc` file is run every time a Korn shell is explicitly invoked.

The file pointed to by the `ENV` variable should contain Korn shell commands.

Instructor Notes:

Purpose — To determine how to define aliases.

Details — The only drawback of an alias is that it is local to the shell that created it. In order to export the alias it must be entered into the **.kshrc** file or the file pointed to by the ENV variable.

Additional Information — The default aliases seem to be built in to the shell.

Transition Statement — Having described what aliases are, let us see how to manipulate the aliases themselves.

ksh Features - Using Aliases

```
$ 11 → alias ll='ls -l'
-rw-r--r-- 1 joe staff 524 Sep 19
           11:31 fleas
-rw-r--r-- 1 joe staff 1455 Jan 23
           17:18 walrus

$ unalias 11 → Remove alias definition

$ 11
ksh: 11: not found
```

© Copyright IBM Corporation 2008

Figure 12-10. ksh Features - Using Aliases

AU1310.0

Notes:

Using aliases

To use an alias, simply invoke the alias name as if it were a command.

It is possible to invoke an alias with parameters, as long as these are significant to the *LAST* command in the alias. For example:

```
alias dir='ls'
dir -l
```

The **-l** will be added to the original command **ls** in the alias **dir**.

Removing aliases

To remove an alias, use the **unalias** command. This causes the current shell to “forget” about the alias. The names of the aliases specified with the **unalias** command will be removed from the alias list.

Instructor Notes:

Purpose — In the previous visual, several aliases were set up. Now let us see how the aliases are used and manipulated.

Details — Remember: if the alias is set via the **.kshrc** file, then it will still be present when you next invoke a shell or log in.

Additional Information —

Transition Statement — We will now finish this unit with a look once more at the history functions of the shell.

ksh Features - History

Last 128 commands are stored in file **\$HOME/.sh_history**

```

$ fc -1
2 cd /home/payroll
3 ls -l
4 mail
5 fc -1

$ r m
No mail for team01

$ r 3
-rw-r--r--    1 joe staff 524      Sep 19 11:31 fleas
-rw-r--r--    1 joe staff 1455     Jan 23 17:18 walrus

```

© Copyright IBM Corporation 2008

Figure 12-11. ksh Features - History

AU1310.0

Notes:

Shell command history

The text of the previous commands entered from a terminal device is stored in a history file, which by default is called **.sh_history** and is stored in the user's **\$HOME** directory.

The **fc -1** command reads this file and allows you to list the last 16 commands entered. Instead of **fc -1** you can use the command **history**.

The **r** command allows you to recall previously entered commands. You can specify the command number (as given by the **history** command) or a text pattern to match against the command name.

The **fc** command allows the last 128 commands in the **.sh_history** file to be examined/modified. The portion of the file to be edited or listed can be selected by number or by giving the first character or characters of the command. If you do not specify an editor program as an argument to the **fc** command the value of the **FCEDIT** variable is used. If the **FCEDIT** variable is not defined, then the **/usr/bin/ed** file is used.

Instructor Notes:

Purpose — This visual explores the shell **history** features.

Details — We saw a few pages back that command lines can be recalled using the Korn shell setting **set -o vi**. There are default facilities provided if the user is not familiar with **vi** or is not particularly interested in having to learn to use the editor.

This file name can be overridden by using the **HISTFILE** variable.

In the examples, the **mail** command can be recalled by using either the line number (4) or by the letter **m**.

Be sure to mention the inherent dangers of using the **r** command, especially without using a number argument. It does not display the retrieved command; it just runs it. If a wrong number is entered, or a string argument which matches a different command than expected, the incorrect command may be executed and cause undesired results. Using the shell command retrieval mechanism (**set -o vi**) is much safer.

While the size of the **.sh_history** file may grow, the number of commands that can be modified/examined in this file using the **fc** command or listed using the **history** alias is determined by the **HISTSIZE** variable. If this variable is not set, the default is 128.

Like the previous form of command line recall (**set -o vi**), the **history** function has a default setting for its preferred editor. This editor will allow you to recall/modify previous commands.

Additional Information —

Transition Statement — Before moving on to the lab exercise, let's review a few checkpoint questions.

Checkpoint

1. Which file would you use to customize your user environment? Why?

2. What do the following variables define on your system?

PS1:

TERM:

PATH:

© Copyright IBM Corporation 2008

Figure 12-12. Checkpoint

AU1310.0

Notes:

Instructor Notes:

Purpose — To test the students' knowledge of this unit's material.

Details —

Checkpoint Solutions

1. Which file would you use to customize your user environment? Why? **\$HOME/.profile as this is the file that overrides the /etc/profile which is the system-defined file.**
2. What do the following variables define on your system?

PS1: primary prompt string (that is, your prompt)

TERM: the terminal type

PATH: the path of directories that is searched, in order to locate an executable

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — Let's do a lab exercise.

Exercise: Customizing the User Environment



© Copyright IBM Corporation 2008

Figure 12-13. Exercise: Customizing the User Environment

AU1310.0

Notes:

After completing the exercise, you will be able to:

- Customize **.profile** and **.kshrc** files
- Set **alias** definitions

Instructor Notes:

Purpose — Describe what students will learn in the lab.

Details —

Additional Information —

Transition Statement — Let's wrap up this unit with a summary of what we have covered.

Unit Summary

- The purpose of the **login profile** was considered in conjunction with the **customization files /etc/profile, /etc/environment, \$HOME/.profile** and **\$HOME/.kshrc**.
- The **shell history mechanism** is one method that can be used to **recall** previous commands.
- **Aliases** can be set up to provide an **alternate name** for commands.

© Copyright IBM Corporation 2008

Figure 12-14. Unit Summary

AU1310.0

Notes:

Instructor Notes:

Purpose — To summarize the key points from this unit.

Details —

Additional Information —

Transition Statement — Let's move on to a new unit.

Unit 13. AIX Utilities, Part I

Estimated Time

00:55

What This Unit Is About

This unit covers a selection of useful commands which can be used to carry out specific tasks.

What You Should Be Able to Do

After completing this unit, students should be able to:

- Use the `find` command to search directories for files with particular characteristics
- Use the `grep` command to search text files for patterns
- Use the `head` and `tail` commands to view specific lines in a file
- Use the `sort` command to manipulate the contents of files
- Use the `telnet` and `ftp` commands to communicate with other hosts

How You Will Check Your Progress

Accountability:

- Student activity
- Checkpoint questions
- Exercises

Unit Objectives

After completing this unit, you should be able to:

- Use the **find** command to search directories for files with particular characteristics
- Use the **grep** command to search text files for patterns
- Use the **head** and **tail** commands to view specific lines in a file
- Use the **sort** command to manipulate the contents of the files
- Use the **telnet** and **ftp** commands to communicate with other hosts

© Copyright IBM Corporation 2008

Figure 13-1. Unit Objectives

AU1310.0

Notes:

Instructor Notes:

Purpose — Introduce the unit's objectives.

Details —

Additional Information —

Transition Statement — Let's start with the `find` command.

find

- **Search** one or more directory structures **for files that meet certain specified criteria**
- **Display the names** of matching files or **execute commands** against those files

```
find path expression
```

© Copyright IBM Corporation 2008

Figure 13-2. `find`

AU1310.0

Notes:

Introduction

The `find` utility is of immense use to anyone who works with files and directories. It can be used to search for misplaced files as well as for performing an action against files that have been located.

Searching for files

An example of this would be to search for `core` files (application program crashes tend to leave a file called `core` in the current directory) and having found them, delete them. This would be useful since it would reclaim wasted disk space!

find command syntax

The syntax of the command is very particular and needs to be expressed in the following way:

```
find <from-where> <searching-for-what> <do-something-to-it>
```

In the syntax shown on the visual, the **expression** is optional.

The **find** command recursively searches the directory tree under each specified path, seeking files that match a search criteria provided in the expression **<searching-for-what>**.

The output from the **find** command depends on the terms specified by the final parameter, **<do-something-to-it..>**.

Instructor Notes:

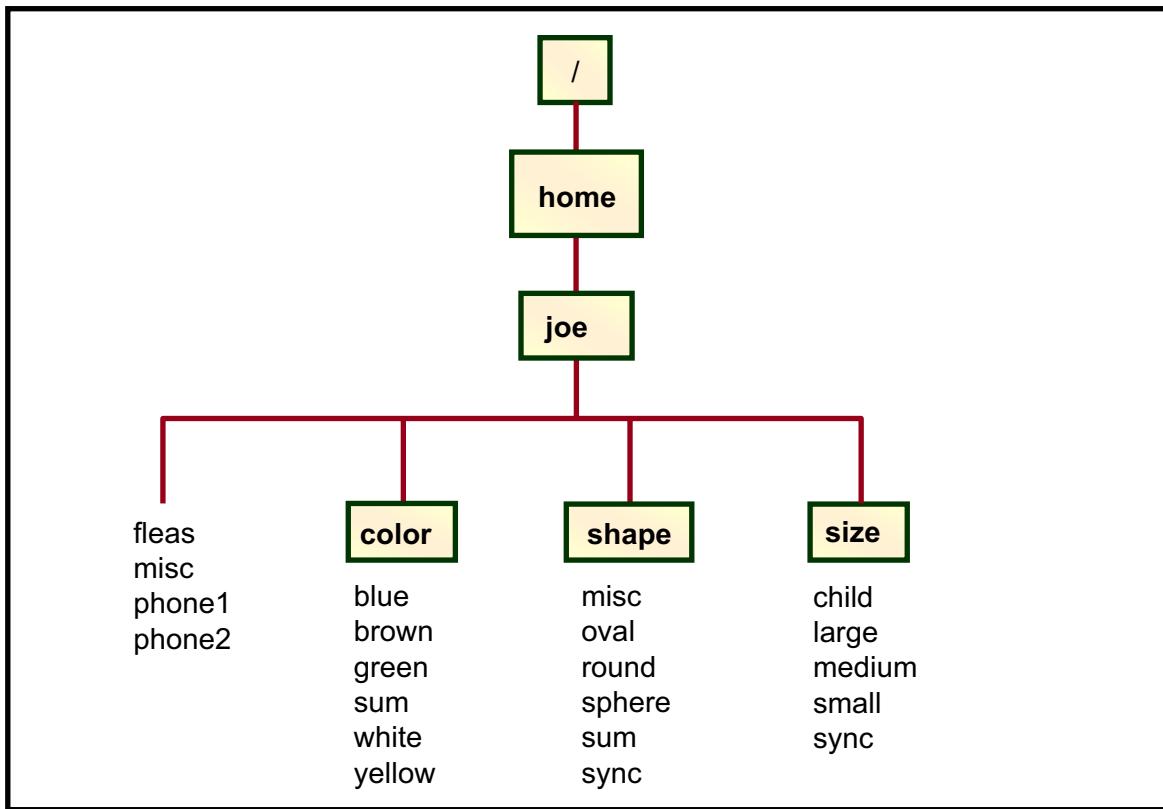
Purpose — Describe the uses and the basic syntax for the `find` command.

Details —

Additional Information — There are very good descriptions of the command and all its options in the online manual pages and Web-based documentation. Note that `find -name` expressions use the same pattern matching conventions as the shell uses for file name substitution.

Transition Statement — We now will take a look at some examples of the `find` command, using the directory structure on the next visual.

Sample Directory Structure



© Copyright IBM Corporation 2008

Figure 13-3. Sample Directory Structure

AU1310.0

Notes:

find examples

This sample directory structure will be used in the examples on the following pages.

Instructor Notes:

Purpose — To show a sample directory structure that will be used with some command examples coming in the next few visuals.

Details — The overall objective is to go through some examples of AIX utilities and show how these can be used using this subtree structure as the example.

Keep this visual up for the examples so students do not have to keep going back and forth.

Additional Information —

Transition Statement — Let's start by working with a utility called `find`.

Using `find`

- Search a [directory structure](#) for files with [certain names](#):

```
$ find . -name sum  
./color/sum  
./shape/sum
```

- On older UNIX systems, you may need to use [-print](#):

```
$ find . -name sum -print  
./color/sum  
./shape/sum
```

© Copyright IBM Corporation 2008

Figure 13-4. Using `find`

AU1310.0

Notes:

Using the `find` command

When searching with `find`, both directories and ordinary files that match the search criteria are listed. The search will search all directories and subdirectories below the path specified in the command.

Note that the `-print` option is the default, and so using it with the command is not required. This was not always the case. In earlier versions of AIX and on other UNIX systems that have not yet implemented the POSIX standard for the `find` command, the `-print` option is required for the result to be displayed or used in a pipe.

Instructor Notes:

Purpose — This is how we use the `find` command.

Details — The first example of the `find` command demonstrates with AIX 5L that when using `find`, it will assume `-print` if you do not provide any of the primaries of `-exec`, `-ok`, or `-print`. This may not be the case with other UNIX systems, so be sure to mention that you may need to specify all the parts to the command in order to get any results.

Discussion Items — A sample question needs to be posed as a reminder that:

1. Directory permissions apply and that you cannot search directories for which you do not have permission.
2. Error messages will appear. A subsidiary question is then how to get rid of these error messages?

You may also find it useful to ask the following: How would the previous search be specified but this time starting from the root directory?

Answer: `find / -name sum -print`

Additional Information —

Transition Statement — The `find` command can also do things to the files or directories that it locates.

Executing Commands with `find`

The `exec` option executes a command on each of the files found.

```
$ find . -name 'm*' -exec ls -l {} \;
-rw-r--r-- 1 joe staff 83 Jan 11 15:55 ./shape/misc
-rw-r--r-- 1 joe staff 21 Jan 11 16:01 ./size/medium
-rw-r--r-- 1 joe staff 38 Jan 11 15:34 ./misc
```

© Copyright IBM Corporation 2008

Figure 13-5. Executing Commands with `find`

AU1310.0

Notes:

Executing commands using results from `find`

The `-exec` option is the non-interactive way to execute commands with `find`. The command following `-exec`, in this example `ls`, is executed for each file name found. The `{}` are used as logical place holders for the matches and `find` replaces the `{}` with the names of the files matched. Note the use of the escaped; `(\;)` to terminate the command that `find` is to execute. This requirement is hard coded within the `find` command and is required for use with the `-exec` and `-ok` options.

The `find` command may also be used with the `-ls` option:

```
$ find . -name 'm' -ls
```

Instructor Notes:

Purpose — To show how a simple command can be executed with `find`.

Details — The syntax will look strange to the student, but that is the way it is. Explain each point to clarify:

<code>find .</code>	Search current directory and below
<code>-name 'm*'</code>	For all files that begin with the letter m
<code>-exec</code>	Execute a command against what has been found
<code>ls -l {}</code>	The command to execute and a command parameter containing the path name
<code>\;</code>	Escaped semicolon for end of command

The result of the command is to produce a long listing of only those files that begin with the letter m from the current directory down.

Additional Information — A question can be used as a reminder of the action of wildcard characters. Remember to point out that these are expanded unless they are quoted.

There is now a `-ls` option with the `find` command since AIX 5L V5.1, which works like `ls -fields`. However the `-exec ls` is a good example for the students. This is worth bearing in mind.

Discussion Items — Ask the following: Why is the '`m*`' in single forward quotes?

Answer: The asterisk is a metacharacter. It must be quoted or it might be substituted by the shell with one or more file names in the target directory, rather than being used by the `find` command to match file recursively under the target directory.

Also ask, "What would have happened if the wildcard was not quoted?"

Answer: It will only locate files called `misc` as that is the only file that the wildcard can expand to in the current directory.

Transition Statement — This form of command execution is non-interactive and suitable for displaying information. There may be occasions where this is not such a good idea and some interaction is required.

Interactive Command Execution

The **ok** option causes **command execution** on an **interactive** basis.

```
$ find . -name m\* -ok rm {} \;
<rm ... ./shape/misc>? y
<rm ... ./size/medium>? y
<rm ... ./misc >? n
```

© Copyright IBM Corporation 2008

Figure 13-6. Interactive Command Execution

AU1310.0

Notes:

Interactive example

Here **find** is also performing a command. This time it will ask before each task is carried out on each file found.

It is a good idea to use the **-ok** option rather than **-exec** if there are not a lot of files that match the search criteria and it may not be desirable to run the command on every found file. It is a lot safer if your pattern is not exactly what you think it is.

Instructor Notes:

Purpose — To see how interaction with the user can be achieved with the `find` command.

Details — The difference between this and the last example is the use of the `-ok` option.

`-exec` used `ls -l` but `-ok` used `rm`.

At each prompt, the user will need to supply a `y` or a `yes` in order for the action to take place, anything else will cause `find` to ignore this file or directory.

Additional Information —

Transition Statement — Having worked through some examples, we will now look at some further options to the `find` command.

Additional Options

-type	f d	ordinary file directory
-size	+n -n n	larger than “n” blocks smaller than “n” blocks equal to “n” blocks
-mtime	+x -x	modified more than “x” days ago modified less than “x” days ago
-perm	onum mode	access permissions match “onum” (that is, 755) access permissions match “mode” (that is, rwx)
-user	user	find files owned by “user”
-o		logical “or”
-newer	file	search for files that are newer than “file”

© Copyright IBM Corporation 2008

Figure 13-7. Additional Options

AU1310.0

Notes:

Other useful options to the `find` command

Complete details to these and many other options to the `find` command are described in the online manuals.

Instructor Notes:

Purpose — These are just some of the many options available to the `find` command.

Details — The options are as follows:

- `type` allow searches for only files or only directories.
- `size` search for files that exactly match a size (-`size 10`) or that are more than a size (-`size +10`) or that are below a certain size (-`size -10`). Size values are expressed in 512-byte blocks.
- `mtime` search for files that have been modified in the time parameter supplied. The times are in days relative to the current day plus 24 hours. The times can be an exact match or older or newer than the time specified.
- `perm` search for files that have a certain permission mask (see `chmod`).
- `o` allow multiple conditions to be matched (find a OR b).
- `newer` searches for files that are newer than the reference file.

Additional Information — See the Web-based documentation to display a wealth of functionality in `find`.

In AIX 5L V5.3 or later, there are new options which allow age criteria in minutes rather than hours. An example is the `-mmin` flag which selects file based on when they were last modified in minutes.

When multiple search options are provided, they are normally ANDed together. This means that they must all be true or the file will not be selected. The `-o` option allows us to also OR our criteria together. Escaped Parenthesis may be used to group the criteria as in traditional Boolean expressions.

Discussion Items — What useful function could you carry out by using the `-mtime` option?

Answer: Backup - find files that are older than x number of days old.

Transition Statement — The next visual depicts `find`'s ability to travel down the tree structure and compares it with the shell. Let's see that comparison.

The Shell versus `find`

Scenario: Starting at current directory find all files that start with c.

```
$ ls c*
```

c1 c2

Shell expands wildcard

```
$ find . -name 'c*''
```

./c1

find expands wildcard

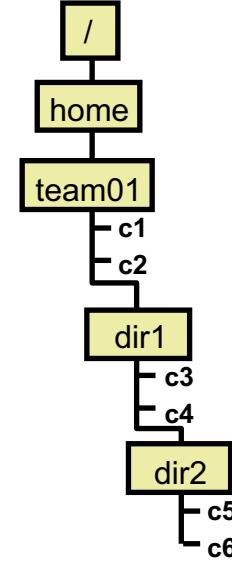
./c2

./dir1/c3

./dir1/c4

./dir1/dir2/c5

./dir1/dir2/c6



© Copyright IBM Corporation 2008

Figure 13-8. The Shell versus `find`

AU1310.0

Notes:

Comparison of shell versus `find`

The most important characteristic of the command `find` is its ability to travel down subdirectories. Normally, the shell provides the argument list to a command. Most commands do not understand directory structures and have to depend on the shell to expand wildcards to directory names. To have the shell actually list all files in all of the subdirectories, the equivalent command would be:

```
$ ls c* */c* */*/c*
```

Instructor Notes:

Purpose — Discuss the difference of the shell gaining control of the command line versus **find**.

Details — In the first example, the shell will expand the wildcard to find all files that start with **c**. Unfortunately, it will only search in the current directory. **ls** does not understand directory structures and relies on the shell to expand the wildcards to directory names. To traverse three directories using **ls**, look at the equivalent command in the notes. This is very tedious to type as well as you would have to know how many subdirectories exist below the current directory in order to get all the files.

Another problem with this is that the shell will NOT match files in directories whose names start with a **.**(dot). Also, if there was another subdirectory under **dir2** it would not be searched.

Using **find** makes sense when subdirectories need to be searched. This is **find**'s most important characteristic.

Additional Information —

Transition Statement — To finish with the **find** command let us review some more examples.

find Examples

```
$ find . -name 's*' -type f -size +2 -exec ls -l {} \;
-rwxr-xr-x 1 joe staff 1512 Jan 11 15:43 ./color/sum
-rwxr-xr-x 1 joe staff 2148 Jan 11 15:57 ./shape/sum

$ find . -perm 644 -mtime +4 -print
./shape/misc

$ find . -name fleas -o -name misc
./misc
./shape/misc
./fleas

$ find / -name 'security' -print 2> errfile
/var/security
/usr/lpp/bos.sysmgt/inst_root/var/security
/usr/lib/security
/etc/security
```

© Copyright IBM Corporation 2008

Figure 13-9. `find` Examples

AU1310.0

Notes:

Examples of `find`

The first example will find, starting from the current directory, all the files that begin with the letter **s** which are ordinary files and are larger than two blocks. Once these have been found, the **ls -l** command will be executed on them.

The second example will find, from the current directory downward, all the files that have their permissions set to 644 and have been modified more than four days ago.

The third example will find all files that are called either **fleas** or **misc**. The search will be started from the current directory downward.

The last example will start the search from the root directory and will pick up all the files that have the string **security** as part of their path name. The path names will be displayed on the screen, however any error messages will be directed to the file **errfile**.

Instructor Notes:

Purpose — The idea is to get the students to think through the syntax of the command one more time. These are fairly complex examples and once understood, the students can claim to be competent with the command.

Details — Get the students to describe what happens here as opposed to you describing each.

The first example reads: Locate all files from the current directory that begin with s and are larger than 1 KB in size; execute a long listing for all files found.

The second example reads: Find and display all files or directories from the current directory that are readable and writable by the owner and read-only for everyone else and have been modified over four days ago.

The third example reads: Find and display all files or directories that are named **fleas** or **misc**.

The fourth example reads: Find and display anything called security from the top of the directory tree, saving all error messages into a file named **errfile** for future reference.

Errors are usually related to AIX file and directory permissions. A normal user will have a large number of error messages in the file **errfile** as a direct consequence of not having access permissions to certain directories on the system.

Additional Information — The second example is a tricky one because usually files have permissions 644. Directories would normally have an equivalent 755. Any directory with 644 permissions is going to be a problem! The find command will try to access the directory to search the content for files meeting the criteria. Without x permission an error message will be displayed.

The visual shows how the command can be coded in AIX 5L V5.3.

Transition Statement — Let's do an exercise to test your understanding of the material presented so far.

Exercise: AIX Utilities (1)



© Copyright IBM Corporation 2008

Figure 13-10. AIX Utilities (1)

AU1310.0

Notes:

After completing the exercise, you will be able to:

- Use the `find` command to find files that meet specific criteria

Instructor Notes:

Purpose — Describe what the students will learn in the lab.

Details —

Additional Information —

Transition Statement — Now that you have completed the first exercise from this unit, we will introduce the `grep` command.

grep

Search for lines matching specified **pattern**

```
grep [options] pattern [file1 file2 ...]
```

Simple
text

Regular
expression

© Copyright IBM Corporation 2008

Figure 13-11. grep

AU1310.0

Notes:

Searching within files

The **grep** command (which stands for Global Regular Expression Parser) searches for the pattern specified and writes each matching line to standard output.

The search can be for simple text, like a string or a name. **grep** can also look for logical constructs, called regular expressions, that use patterns and wildcards to symbolize something special in the text, for example, only lines that start with an uppercase T.

The command displays the name of the file containing the matched line, if more than one file is specified for the search.

Instructor Notes:

Purpose — Introduce the `grep` command.

Details — The `grep` command is used to locate information in a file. It takes horizontal slices, or lines, out of a file if a match is found.

The search can be for simple text, like a string or a name. `grep` can also look for logical constructs, called regular expressions, that use patterns and wildcards to symbolize something special in the text, for example lines that start with an uppercase T and no others. A regular expression defines a set of patterns that a sequence of characters may match.

`grep` can search multiple files for these patterns.

To use `grep`, you must supply a string or pattern as well as a file name. The file name is optional. You can ask the students what would happen if the file name is omitted.

(Answer: `grep` will read STDIN and display output to STDOUT when a match for the supplied pattern is found.)

Additional Information — Read the Web-based documentation for details about the many options and for an overview of regular expressions.

Transition Statement — We shall now take a look at how the `grep` command works.

grep Sample Data Files

phone1:

As of: 1/31/2000			
Anatole			389-8200
Avis	Betty	817	422-8345
Baker	John		656-4333
Computer Room	CE phone		689-5790
Dade Travel	Sue		422-5690
Hotline	HW	800	322-4500

phone2:

As of: 2/15/2000			
Anatole			389-8200
Avis	Betty	817	422-8345
Baker	John		656-4333
Computer Room	CE phone		592-5712
Dade Travel	Sue		422-5690
Hotline	HW	800	322-4500

© Copyright IBM Corporation 2008

Figure 13-12. grep Sample Data Files

AU1310.0

Notes:

grep examples

This visual shows the sample files used to illustrate the examples of **grep** that follow on the next visual.

Instructor Notes:

Purpose — This visual presents the sample files used to illustrate the examples of `grep` that follow.

Details — Keep this visual up throughout examples for students' reference.

Additional Information —

Transition Statement — We now shall look at how to use the `grep` utility.

Basic grep

```
$ grep 800 phone1
```

Hotline

HW 800 322-4500

```
$ grep 800 phone*
```

phone1:Hotline

HW 800 322-4500

phone2:Hotline

HW 800 322-4500

© Copyright IBM Corporation 2008

Figure 13-13. Basic grep

AU1310.0

Notes:

grep examples

The first example will search a file called **phone1** for any lines containing the sequence 800.

In the second example, a wildcard is used that will search all files that start with **phone** for the pattern.

Instructor Notes:

Purpose — Describe how to use the `grep` utility.

Details — `grep` will display lines from files that contain the pattern. All other lines will not be displayed. `grep` will be used most often as part of a filter or as part of an AIX pipe. Its function will be to extract certain information from STDIN for further processing.

The examples illustrate the basic use of `grep`.

The difference between the first and second example is that in the second example, a shell (file name substitution) wildcard is used that will search ALL matched files for the pattern.

By default, this time the file names will also be displayed.

Additional Information —

Transition Statement — Now that we have described the basic usage of `grep`, let's take a look at regular expressions.

grep with Regular Expressions

```
grep 'regular_expression' file
```

Valid metacharacters:

.	Any single character
*	Zero or more occurrences of the preceding character
[aA]	Enumeration: a or A
[a-f]	Any ONE of the characters in the range of a through f
^a	Any lines that start with a
z\$	Any lines that end with a z

© Copyright IBM Corporation 2008

Figure 13-14. grep with Regular Expressions

AU1310.0

Notes:

Introduction

The **grep** command uses patterns to represent text in a file. These patterns are called *regular expressions*.

Regular expressions

Some of these patterns represent unique characteristics of a line, for example, lines that start with the letter **T** or lines that have the letters **xyz** somewhere in the line. **grep** uses its own set of metacharacters. These are slightly different from those used by **find** and the shell.

When ***** is used with the **grep** command to specify a regular expression, it will match zero or more occurrences of the previous character. If you want to use it like a wildcard, it should be preceded by a dot, which means any single character.

Metacharacters

The **grep** command metacharacters are:

- . any one character
- * a wildcard applied to the previous character only, indicating whether or not it is repeated
- [**x-z**] a range of characters between **x** and **z**
- ^a** indicates a search for a line starting with the character **a**
- z\$** indicates a search for a line ending with the character **z**

With these regular expressions it should be possible to describe any string and to search for any combination of characters.

The following is a chart which compares **grep**'s metacharacters to the shells metacharacters:

grep	grep Interpretation	Shell	Shell Interpretation
^	begins a line	^	old Bourne pipe symbol
\$	ends a line	\$	variable
.	single character	?	single character
.*	multicharacter	*	multicharacter
[-]	single character	[-]	single character

NOTE: Patterns with metacharacters should be in single quotes (' '), so that the shell will leave it alone.

Instructor Notes:

Purpose — This visual explains the concepts of patterns.

Details —

Additional Information —

Transition Statement — Now, let's try out some of these regular expressions.

grep Examples

```
$ ps -ef | grep team01
team01 10524 13126 0 09:27:45 pts/1 0:00 -ksh
```

```
$ grep '^B' phone1      ^: Start
Baker           John          656-4333
```

```
$ grep '5$' phone1      $: End
Avis            Betty         817 422-8345
```

```
$ grep '^([DH])' phone1 [DH]: Enumeration
Dade Travel     Sue          422-5690
Hotline          HW           800 322-4500
```

```
$ grep '^A.*0$' phone1   .*: Zero or more occurrences of any single character
As of: 1/31/2000
Anatole          389-8200
```

© Copyright IBM Corporation 2008

Figure 13-15. **grep** Examples

AU1310.0

Notes:

grep examples

In the first example, **grep** reads from standard input and filters all the processes that have been started by **team01**.

In the next example, **grep** prints all the lines from the **phone1** file that begin with the letter **B**.

The third example prints all the lines that end with the number **5**.

The next example prints all the lines that start either with the letter **D** or **H**.

The last example shows the meaning of the regular expression **.***. All the lines are printed that start with an **A**, followed by any characters that end with the number **0**.

Instructor Notes:

Purpose — Introduce regular expression syntax.

Details — Explain the different examples.

Additional Information — Beware of searching for patterns of text that contain these metacharacters as you will have to escape (\) them.

Transition Statement — These are the pattern matching tools that are at your disposal. Most other utilities, like **vi**, will use the same syntax for locating strings of text.

grep has a number of options, some of these make searching for patterns easier.

grep Options

-v	print lines that do not match
-c	print only a count of matching lines
-l	print only the names of the files with matching lines
-n	number the matching lines
-i	ignore the case of letters when making comparisons
-w	do a whole word search

© Copyright IBM Corporation 2008

Figure 13-16. grep Options

AU1310.0

Notes:

grep options

The visual shows different grep options.

Instructor Notes:

Purpose — Explain the most important `grep` options.

Details —

Additional Information — AIX 5L V5.3 introduced a new option that allows recursive searches. The `-r` flag searches directories recursively. By default, links to directories are followed. `-R` searches directories recursively but links to directories are not followed.

Transition Statement — `grep` has other forms. These are optimized for specific tasks. Let's take a look at these.

Other grep Commands

•fgrep

fast grep: Only fixed strings, no regular expressions

```
$ fgrep 'HW' phone1
```

Hotline	HW	800 322-4500
---------	----	--------------

•egrep

Extended grep: Allows for multiple patterns

```
$ egrep '800|817' phone1
```

Avis	Betty	817 422-8345
Hotline	HW	800 322-4500

© Copyright IBM Corporation 2008

Figure 13-17. Other grep Commands

AU1310.0

Notes:

Introduction

grep is a useful tool that extracts text from a data stream such as a file. There are occasions when extra features are required. These could be either performance related or even requiring further functionality not normally associated with the basic **grep** command. These are accomplished with the **egrep** and **fgrep** commands.

Other search commands

egrep, or extended grep, does everything **grep** can do plus it allows OR searches using the pipe (|) character is used to separate the patterns to be searched for. **egrep** is slightly slower than normal **grep**.

fgrep is slightly faster because there is no interpretation that must take place first. **fgrep** only performs string searches. No regular expressions are allowed.

Note that **grep**, **egrep** and **fgrep** have the same i-node and will work different due to the command.

```
$ cd /usr/bin  
$ ls -lai *grep  
6235 -r-xr-xr-x 3 bin bin 19174 Sep 16 02:49 egrep  
6235 -r-xr-xr-x 3 bin bin 19174 Sep 16 02:49 fgrep  
6235 -r-xr-xr-x 3 bin bin 19174 Sep 16 02:49 grep
```

grep = egrep = fgrep

In AIX, the commands **grep**, **fgrep**, and **egrep** all the same physical executable, with three different i-node references (hard links).

```
$ ls -li /usr/bin/*grep  
94405 -r-xr-xr-x 3 bin bin 34232 Oct 30 12:58 /usr/bin/egrep  
94405 -r-xr-xr-x 3 bin bin 34232 Oct 30 12:58 /usr/bin/fgrep  
94405 -r-xr-xr-x 3 bin bin 34232 Oct 30 12:58 /usr/bin/grep  
$
```

Instructor Notes:

Purpose — Explain the use of **egrep** and **fgrep**.

Details —

Discussion Items — How could you search for the word **the** or for the word **fleas** in the **fleas** file?

Answer: **\$ egrep "the|fleas" fleas**

Additional Information —

Transition Statement — Let's do a short activity.

Activity: grep Command



© Copyright IBM Corporation 2008

Figure 13-18. Activity: `grep` Command

AU1310.0

Notes:

Activity

- ___ 1. Log in to the system.
- ___ 2. List all processes that contain the word **root**.

- ___ 3. The file **/etc/passwd** stores all AIX users. Using **grep**, list all lines from this file, that start with **t**. Write down the command:

- ___ 4. Change the last command and print out only the count of matching lines.
- ___ 5. List all lines from **/etc/passwd** that do not start with a **t**.
- ___ 6. The third field in each line contains the **user ID**. List all users that have a user ID between 200 and 299.
- ___ 7. Using **find** and **grep -l** list all the file names below **/home** that contain the string **MAILMSG**. Redirect standard error to **/dev/null**.

Instructor Notes:

Purpose — To test the students' knowledge of the material presented so far.

Details — Give the students some time to work on this activity. Here are the commands:

2. `ps -ef | grep root`
3. `grep '^t' /etc/passwd`
4. `grep -c '^t' /etc/passwd`
5. `grep -v '^t' /etc/passwd`
6. `grep '^.*.*:2[0-9][0-9]:' /etc/passwd`
7. `find /home -exec grep -l 'MAILMSG' {} \; 2>/dev/null`

Additional Information —

Transition Statement — Let's introduce the `sort` command.

sort Command

The **sort** command sorts lines and writes the result to standard output:

```
$ sort [-t delimiter] [+field[.column]] [options]
```

Options:

-d	Sorts in dictionary order. Only letters, digits and spaces are considered in comparisons.
-r	Reverses the order of the specified sort.
-n	Sorts numeric fields in arithmetic value.
-t	Tells sort what character separates fields.

© Copyright IBM Corporation 2008

Figure 13-19. **sort** Command

AU1310.0

Notes:

Introduction

The **sort** command is used to sort the content of a file or STDIN before it is sent to STDOUT. This ensures that the output is in the right order.

The processing uses either dictionary or ASCII order in sorting the data. This can be controlled by the use of options (ASCII is the default).

Changing the delimiter

sort uses a tab or a space as the default delimiter between fields. To specify a delimiter with **sort** use the **-t** option. The **-t** option tells **sort** what character separates fields. This is often a **:**, **\t** (tab), or **\n** (new line) character.

Instructor Notes:

Purpose — To introduce the `sort` command.

Details — The `sort` command works by reading from STDIN or file arguments and processing the data before sending it to STDOUT.

`sort` uses a tab or a space as the default delimiter between fields. If a combination of tabs and spaces is used inconsistently throughout the file being sorted, the tabs and spaces themselves will be sorted producing what looks like an incorrect sort.

For example:

With this file:

Sort +1 produces:

file<space>file4	file<tab>file1
file<space>file7	file<tab>file8
file<tab>file8	file<space>file3
file<tab>file1	file<space>file4
file<space>file3	file<space>file7

Additional Information — For further details of the syntax of the `sort` command, please refer to the manual pages of the online documentation.

Be careful! The command line options are unique to this command. For example, in the `cut` command, the delimiter option is `-d`. In `sort` it is `-t`.

Transition Statement — Having described the workings of the `sort` command, let us take a look at some examples of using the `sort` command.

sort Examples

```
$ cat animals
```

```
dog.2  
cat.4  
elephant.10  
rabbit.7
```

```
$ sort animals
```

```
cat.4  
dog.2  
elephant.10  
rabbit.7
```

Default sort order

```
$ cat animals | sort +0.1
```

```
rabbit.7  
cat.4  
elephant.10  
dog.2
```

Sort by second character

```
$ cat animals | sort -t. -n +1
```

```
dog.2  
cat.4  
rabbit.7  
elephant.10
```

-t: Delimiter ".
-n: Numerical order
+1: Second field

© Copyright IBM Corporation 2008

Figure 13-20. sort Examples

AU1310.0

Notes:

Examples

This visual shows different ways the **sort** command can be used.

Instructor Notes:

Purpose — To demonstrate the different ways the `sort` command can be used.

Details — The sample data file used in this example is a file called **animals**. This file is not in the students' directories. Use `/etc/passwd` if an example is required.

The first example simply sorts the file animals with the default sort order.

The second example will sort the file by the second character of the first word.

The third example specifies that the word delimiter is a dot (.) and that sorting should take place on the second field in numerical order.

Additional Information —

Transition Statement — The `sort` command is very good at ordering the output of a command. We shall now take a look at two commands which will enable us to view only parts of a file.

head and tail Commands

The **head** command can be used to view the **first few lines** of a file or files.

```
head [-number_of_lines] file(s)
```

```
$ head -5 myfile
```

```
$ ls -l | head -12
```

The **tail** command writes a file to standard output, **beginning at a specified point**.

```
tail [-number_of_lines | +starting_line_number] file(s)
```

```
$ tail -20 file
```

```
$ tail +20 file
```

© Copyright IBM Corporation 2008

Figure 13-21. **head** and **tail** Commands

AU1310.0

Notes:

Extracting lines from a file

The **head** and **tail** commands can be used to extract a number of lines from the top or bottom of a file.

head command

The **head** command is used to display the first few lines of a file. The option to the **head** command specifies the number of lines to display. Ten lines is the default.

tail command

The **tail** command is used to display the last few lines of a file. If no options are specified, the last 10 lines will be displayed. The options to the **tail** command can be used with either a positive or a negative number.

-number_of_lines	specifies the number of lines to read beginning from the end of the file
+starting_line_number	indicates displaying the file beginning at the specified number from the top right through to the end

Tailing an active file

The **tail -f** command can be used to monitor the growth of a file being written by another process. The **-f** option causes the **tail** command to continue to read additional lines from the input file as they become available. For example:

```
tail -f accounts
```

will display the last 10 lines of the **accounts** file. The **tail** command continues to display lines as they are added to the accounts file. The display continues until **<Ctrl-c>** is pressed.

Instructor Notes:

Purpose — To demonstrate the use of the `head` and `tail` commands.

Details — Only explain the use of these commands in their simplest of forms as shown.

The first `head` example shows how the first five lines can be extracted in a file.

The second example illustrates how the output of one command (in this case the `ls` command) can be piped through to the `head` command which will output to the screen only the specified number of lines.

The `tail` examples illustrate the differences between the - and the + symbols: from the top

- indicates the number of lines from the bottom of the file
- + indicates display the lines starting at the number specified from the top of the file

Additional Information — To obtain more information about both commands, use the AIX online documentation or the `man` command.

Transition Statement — Let's look at connecting to remote hosts.

telnet: Login to Remote Hosts

Use the **telnet** command to login to remote hosts.

Example:

```
$ telnet miami
Trying ...
Connected to miami
...
```

```
AIX Version 5
(C) Copyright by IBM and others 1982, 1996
login: team01
```

© Copyright IBM Corporation 2008

Figure 13-22. **telnet**: Login to Remote Hosts

AU1310.0

Notes:

telnet command

The **telnet (tn)** command allows a user to log in on remote systems. This command works in heterogeneous TCP/IP networks and is available on all UNIX systems and many other operating systems.

To log in, you must supply a user name (must exist on the remote system) and normally a password. After a successful login, a shell is started on the remote system.

Instructor Notes:

Purpose — Introduce the student to remote host access.

Details — While `tn` and `telnet` are hard links to the same binary executable program, there are some functional differences. Firstly, invoking `tn` causes the interactive prompt to show `tn>`. Also, to enter the interactive prompt from within a connected session is `<Ctrl-t>` for `tn` and `<Ctrl-]>` for `telnet`.

Additional Information —

Transition Statement — Let's look at transferring files between hosts.

ftp: Transfer Files Between Hosts

Use the **ftp** command to transfer files between hosts.

Example:

```
$ ftp miami  
Connected to miami  
220 FTP server ready  
Name (miami: team01): team05  
Password required for team05.  
Password:  
230 User team05 logged in.
```

ftp>

ftp prompt waiting for subcommands

© Copyright IBM Corporation 2008

Figure 13-23. **ftp**: Transfers Files Between Hosts

AU1310.0

Notes:

Transferring files using **ftp**

To copy files in a network, the **ftp** command can be used. Like the **tn** command, **ftp** can be used in heterogeneous TCP/IP networks.

You must specify a user name that must exist on the remote system. After a successful authentication, an **ftp** prompt is shown where you specify **ftp** subcommands. The most important subcommands are shown on the next visual.

Instructor Notes:

Purpose — Introduce the student to file transfers using `ftp`.

Details —

Additional Information —

Transition Statement — Let's take a closer look at some of the `ftp` subcommands.

ftp Subcommands

The most important **ftp** subcommands are:

```

ftp>      pwd
ftp>      cd RemoteDir
ftp>      dir          (or)      ls -l
ftp>      get RemoteFile [LocalFile]
ftp>      put LocalFile      [RemoteFile]
ftp>      help [subcommand]
ftp>      quit

```

© Copyright IBM Corporation 2008

Figure 13-24. **ftp** Subcommands

AU1310.0

Notes:

Controlling **ftp**

All **ftp** subcommands must be supplied in the **ftp** prompt (**ftp>**). Here are some examples:

```

ftp> get file1 /tmp/file1
200 PORT command successful
150 Opening data connection for file1 (179 bytes)
226 Transfer complete
ftp> put /subdir1/test1.c c_test.c
200 PORT command successful
150 Opening data connection for c_test.c(201 bytes)
226 Transfer complete
ftp> quit
221 Goodbye

```

Instructor Notes:

Purpose — More details on the subcommands of `ftp`.

Details —

Additional Information —

Transition Statement — Let's look at the utilities that allow non-interactive remote execution.

rexec, rsh: Non-interactive Remote Execution

Use **rexec** or **rsh** to execute a remote command non-interactively.

```
$ rexec sys1 date
Name (sys1:team01): team01
Password (sys1:team01):
Fri Nov 23 14:38:23 EST 2007

$ rsh sys1 date
Fri Nov 23 14:38:30 EST 2007

$
```

- **rsh** requires either **/etc/hosts.equiv** or **\$HOME/.rhosts** config file be set up correctly
- **rexec** can automate logins using **\$HOME/.netrc**

© Copyright IBM Corporation 2008

Figure 13-25. rexec, rsh: Non-interactive Remote Execution

AU1310.0

Notes:

The **rexec** command

The **rexec** command executes a command on the specified remote machine. The host parameter specifies the name of the host where the command is to be executed. The command parameter specifies the command, including any flags or parameters, to be executed on the remote host.

The **rexec** command provides an automatic login feature by checking for a **\$HOME/.netrc** file that contains the user name and password to use at the remote host. If such an entry is not found, the **rexec** command prompts for a valid user name and password for the remote host.

rexec cannot handle commands that use a full screen such as **vi** or graphical applications.

The **rsh** command

The **rsh** command also executes the specified command on the remote host, very similarly to **rexec**.

The remote host allows access only if at least one of the following conditions is satisfied:

- The local user ID is not the root user, and the name of the local host is listed as an equivalent host in the remote **/etc/hosts.equiv** file.
- If either the local user ID is the root user or the check of **/etc/hosts.equiv** is unsuccessful, the remote user's home directory must contain a **\$HOME/.rhosts** file that lists the local host and user name.

Instructor Notes:

Purpose — Describe rexec and rlogin and how they work.

Details —

Additional Information — These commands are very insecure, as they transmit either the password in cleartext, or in the case of rsh, not at all, and can be used by crackers to gain access to AIX.

Transition Statement — Let's look at the secure shell utilities provided by OpenSSH and OpenSSL.

Secure Shell Utilities (OpenSSH)

- Provide secure command replacements for **telnet**, **ftp**, **rexec**, **rlogin**, **rcp**, and **rsh**.
 - Data is encrypted over the network
 - Remote host is verified before connection is allowed
- Commands available are:
 - **ssh**: Remote login and remote execution
 - **scp**: Remote copy
 - **sftp**: Encrypted FTP

```
$ ssh team01@sys1
The authenticity of host 'sys1 (192.168.1.1)' can't be established.
RSA key fingerprint is 21:b0:91:cb:6b:c7:47:7d:96:8d:73:43:44:e8:e3:8d.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'sys1,192.168.1.1' (RSA) to the list of known
hosts.
team01's password:
$
```

© Copyright IBM Corporation 2008

Figure 13-26. Secure Shell Utilities (OpenSSH)

AU1310.0

Notes:

The Secure Shell Utilities

One of the issues that arises when using the traditional TCP/IP utilities is security. When using remote login programs like **telnet** or remote data transfer programs like **ftp**, the data is transmitted over the network in the clear. Thus, anyone with a “sniffer” program can view all the raw data. If this data contains sensitive information (like passwords), they can be easily discovered.

The OpenSSH utilities allow for secure communications between machines. The data stream is encrypted from the client to the server, thereby making it very difficult for a “sniffer” program to see the actual data. These utilities also provide a way that the remote machine is the correct one, and that it has not been hijacked or spoofed by a cracker.

The utilities provided by OpenSSH are:

- **ssh**: Remote login and non-interactive remote command execution

- **scp:** Remote file copy

AIX also provides encryption support for the **ftp** command (via the **-s** option).

If these utilities may not be available by default; ask your system administrator to install them if necessary.

Instructor Notes:

Purpose — Discuss the OpenSSH utilities.

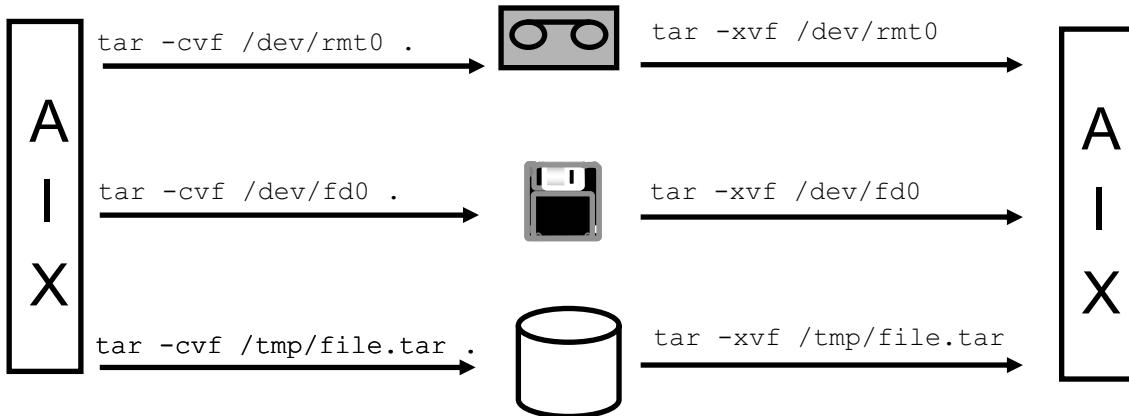
Details — The OpenSSH utilities used to be included as part of the AIX Toolbox for Linux. As of AIX 6.1, they are now included as part of the AIX Expansion Pack. They may not be installed by default.

Additional Information — The **ftp** command has been enhanced to use the OpenSSL libraries for data encryption. This introduces a new flag, **-s**. For this to work, the OpenSSL libraries need to be installed. They are available on the AIX Expansion Pack CD. For more information on SSL support for FTP, please consult the *IBM AIX Version 6.1 Differences Guide*.

Transition Statement — Let's move on to look at archiving files.

tar: Backup and Restore Files

tar (tape archiver) saves files **recursively** and stores them as **one archive file**.



To show the content of the archive file:

`tar -tvf /dev/rmt0` (or `/dev/fd0`)

© Copyright IBM Corporation 2008

Figure 13-27. **tar**: Backup and Restore Files

AU1310.0

Notes:

The **tar** command

The **tar** command saves files and directories in an archive file. In the examples on the visual the archive file is written to a tape file (`/dev/rmt0`), a diskette file (`/dev/fd0`) or to the disk (`/tmp/file1.tar`).

If you specify the dot (.) as shown on the visual the files are saved relatively which allows you to restore the files in a new directory.

The **tar** options are:

- c create
- t table of contents
- v verbose
- f file (archive file name)
- r extend archive
- x extract

Instructor Notes:

Purpose — Introduce the student to the `tar` command.

Details —

Additional Information —

Transition Statement — Before we go to the lab, a few checkpoint questions.

Checkpoint

1. Which commands would you use to locate all the files in your system that began with the string "smit"?

2. What is the following command doing?

```
$ ps -ef | grep -w root | grep -w netscape
```

3. Indicate what the following command is doing:

```
$ ls -l /home | egrep 'txt$ | team01$' | sort -r +7 | tail +4 | head -5
```

© Copyright IBM Corporation 2008

Figure 13-28. Checkpoint

AU1310.0

Notes:

Instructor Notes:

Purpose — Test the students' knowledge on this unit's material before going to lab.

Details —

Checkpoint Solutions

- Which commands would you use to locate all the files in your system that began with the string "smit"?

find / -name 'smit*'

- What is the following command doing?

\$ ps -ef | grep -w root | grep -w netscape

List all processes which have both the root and netscape strings on their ps -ef report lines.

- Indicate what the following command is doing:

\$ ls -l /home | egrep 'txt\$ | team01\$' | sort -r +7 | tail +4 | head -5

A long listing will be carried out from the /home directory, and lines ending with txt or team01 will be picked out and piped through and sorted with the following results:

Once the sort is completed, the output will be piped to tail which will only write line 4 and beyond through the pipe to the head command which will only write to the screen the first 5 lines that it receives (lines 4 through 8 of the sorted file).

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — Let's move on to a lab exercise so you practice this unit's material.

Exercise: AIX Utilities (2)



© Copyright IBM Corporation 2008

Figure 13-29. Exercise: AIX Utilities (2)

AU1310.0

Notes:

After completing this exercise, you will be able to:

- Search text files for specific patterns
- Extract specific fields within a file
- Sort lines in a file
- Use the `head` and `tail` commands

Instructor Notes:

Purpose — To introduce the next lab exercise.

Details — Describe what the students will learn with this exercise.

Additional Information —

Transition Statement — Let's summarize the key points that this unit has covered.

Unit Summary

- The **find** command is used to recursively **search** directories for **files** with particular characteristics.
- The **grep** command is used to **select entire lines** containing a particular pattern.
- The **head** and **tail** commands are used to **view specific lines** in a file.
- The **sort** command **sorts the contents of the file** by the options specified.
- Files from a DOS environment **can be manipulated** in AIX using the following commands: **dosread**, **doswrite**, and **dosdel**.

© Copyright IBM Corporation 2008

Figure 13-30. Unit Summary

AU1310.0

Notes:

Instructor Notes:

Purpose — To summarize this unit.

Details —

Additional Information —

Transition Statement — Let's move on to a new unit.

Unit 14. AIX Utilities, Part II

Estimated Time

00:55

What This Unit Is About

This unit discusses additional helpful utilities that can be used in the AIX environment.

What You Should Be Able to Do

After completing this unit, students should be able to:

- Use the `xargs` command
- Use the `-links` option with `find`
- Use `which`, `whereis`, and `whence` to determine where a command is located
- Determine the type of a file using the `file` command
- Use `diff` and `cmp` to compare files
- Use `dirmp` to compare directories
- Compress files to save space
- Display non-printable characters in files and directories

How You Will Check Your Progress

Accountability:

- Checkpoint questions
- Exercise

Unit Objectives

After completing this unit, you should be able to:

- Use the **xargs** command
- Use the **-links** option with **find**
- Use **which**, **whereis**, and **whence** commands
- Determine the type of a file using the **file** command
- Use **diff**, **cmp**, and **dirncmp** to compare files and directories
- **Compress files** to save space
- Display **non-printable** characters in files and directories

© Copyright IBM Corporation 2008

Figure 14-1. Unit Objectives

AU1310.0

Notes:

Instructor Notes:

Purpose — To introduce the students to the unit's objectives.

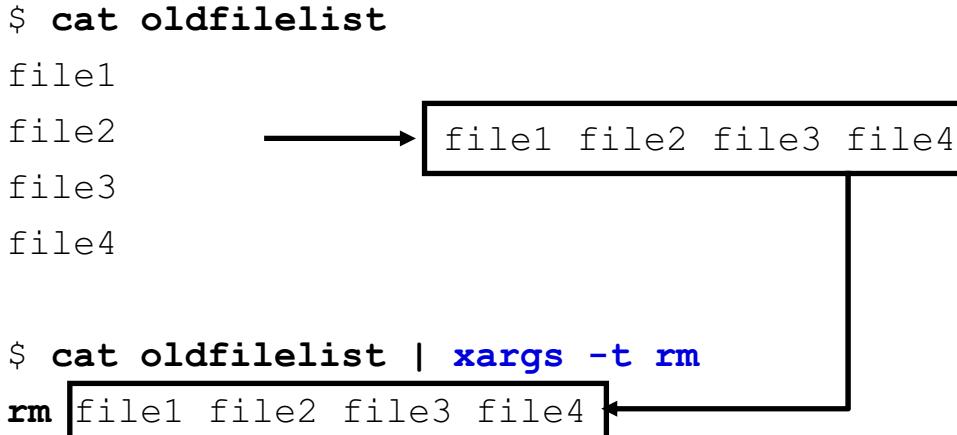
Details —

Additional Information —

Transition Statement — Let's start this unit with a look at the `xarg` utility.

xargs

Reads a **group of arguments from stdin**; runs an AIX command with that group of arguments:



© Copyright IBM Corporation 2008

Figure 14-2. **xargs**

AU1310.0

Notes:

Introduction

This command is one of the best commands you can utilize to execute commands and programs more efficiently and effectively. **xargs** reads arguments one line at a time from STDIN and assembles as many of them as will fit into one command line. It keeps reading arguments and running the command until it runs out of arguments.

xargs command

In the example, **oldfilelist** contains a list of files that need to be removed from the system. Rather than invoking the **rm** command multiple times, or invoking **find** with wildcards to select just the files that should be removed, **cat** passes **xargs** the list of files and allows **xargs** to pass them to **rm**. **xargs** translates information coming from STDIN and will pass each one of those parameters to the parameter line following the subsequent command.

The **-t** flag is optional. It enables trace mode and echoes the constructed command line to STDERR before running, allowing you to see exactly what **xargs** has assembled.

Instructor Notes:

Purpose — Describe the purpose of `xargs`.

Details — Before we cover the example, let's discuss the behavior of certain types of commands. Some commands are smart enough to take information from the parameter line and execute it. Should the parameter line be empty, those smart commands can take input from STDIN, which is generally a pipe. However, some commands are not that smart and cannot accept information from a pipe. `rm` is one of those commands. This is a case where `xargs` is used to translate the information coming from the pipe and will translate each one of those parameters to the parameter line following the `rm` command. The value of the `-t` flag shows you how `xargs` is assembling the command. Once you learn to trust `xargs`, you probably will not use this flag.

The example shows you a very quick way to take a list of diverse files and feed them to the `rm` command.

`xargs` is also a more efficient way to run a command. For example, if you use the `for` loop to cause the `rm` command to be invoked for every single file, then you would be causing a subprocess to be invoked for every single line. `xargs` is smart enough to know how long of a parameter line it can produce and will actually feed many parameters to the `rm` command before `rm` gets invoked the first time; therefore, with `xargs` the command gets invoked much less often, which is a more efficient way to invoke the command.

We have introduced `xargs` first in this unit, as it will be used with some of the commands that follow.

Additional Information —

Transition Statement — Let's take a look at another example of `xargs`.

xargs Examples

```
$ ls > printlist
$ vi printlist
file1
file2
file3
...
file10
$ xargs -t qprt < printlist
qprt file1 file2 file3 file4 file5 ... file10
```

```
$ ls | xargs -t -I {} mv {} {}.old
mv apple apple.old
mv banana banana.old
mv carrot carrot.old
```

© Copyright IBM Corporation 2008

Figure 14-3. **xargs** Examples

AU1310.0

Notes:

Examples using the **xargs** command

In the first example, you want to print a large number of files in a directory. First, redirect the output of the **ls** command to a file and edit the file to remove any files you do not want printed. Pass it to **xargs**. **xargs** will run one or more **qprt** commands, each with a group of arguments until it has read every line in the file.

In the second example, the **{}** symbols allow you to insert file names in the middle of a command line. This command sequence renames all files in the current directory by adding **.old** to the end of each name. The **-I** tag tells **xargs** to insert each line of the **ls** directory listing where the **{}** symbols appear. The **{}** symbols act as a place holder.

Instructor Notes:

Purpose — Cover additional examples of `xargs`.

Details — This is a very efficient way to print a large number of files in a directory. By passing the information to `xargs` to handle, multiple parameters can be passed to `lp` rather than invoking an `lp` command per file.

Additional Information —

Transition Statement — Let's see how we can combine `xargs`, `find`, and `grep`.

xargs, find, and grep

```
$ find . -type f -mtime +30 | xargs -t rm
rm ./file1 ./file2 ./file3 ./file4
```

```
$ find . -type f | xargs -t grep -l Hello
grep -l Hello ./file5 ./file7 ./file10
./file7
```

```
$ find . -type f | xargs -t grep abc
grep abc ./file1 ./file2 ./file3 ./file4
```

© Copyright IBM Corporation 2008

Figure 14-4. **xargs**, **find**, and **grep**

AU1310.0

Notes:

Combining commands

The first example will find all files starting with the current directory whose modification date is older than 30 days and remove them.

The **find** command used without **xargs** would be:

```
$ find . -type f -mtime +30 -exec rm {} \;
```

Using **xargs** is more efficient in that it will pass multiple parameters to **rm**. Also, the syntax when using **xargs** may be easier to remember than the syntax of the full **find** command.

In the second example, **find** gets a list of files in the current directory and passes that list to **xargs**, which will call on **grep** to look inside the files to find which files contain the word **Hello**.

In the third example, **find** gets a list of all files in the current directory and passes the list to **xargs**, which calls **grep** to look inside the files for any occurrence of the string **abc**. Performing the same task without **xargs** would be:

```
$ find . -type f -exec grep abc {} \;
```

This would result in a list of all lines that had the matching string, but not which file the match occurred in.

Instructor Notes:

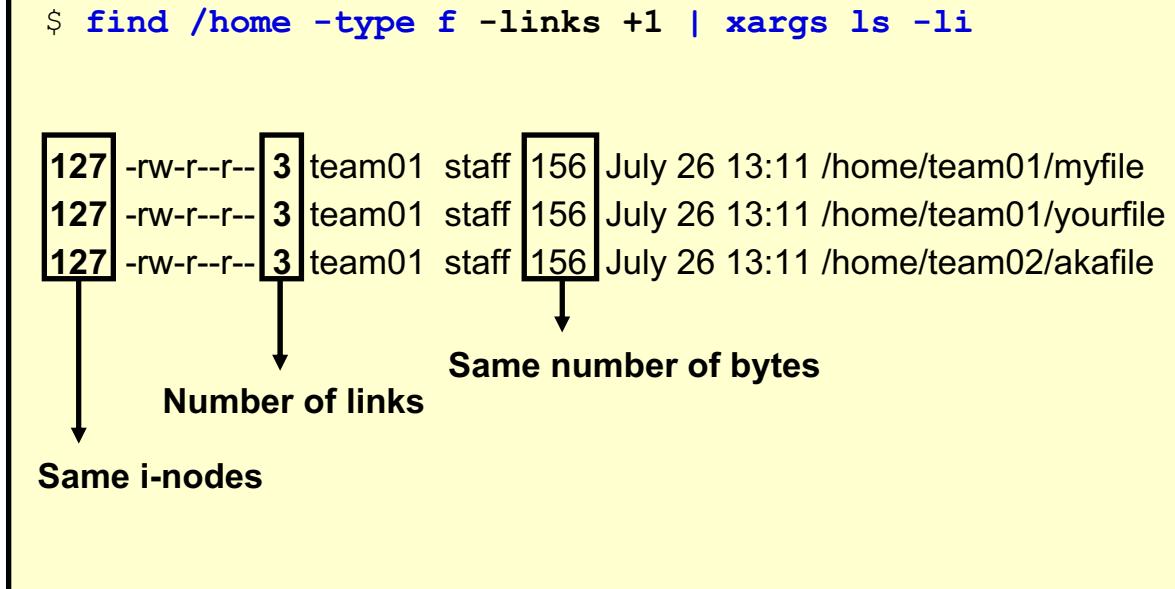
Purpose — Explain how we can use `find` and `xargs` in a pipe to execute many different kinds of commands against a collection of files identified using the find recursive search criteria.

Details —

Additional Information —

Transition Statement — Let's take a look at another expression that can be used with `find`.

The **-links** Option with **find**



© Copyright IBM Corporation 2008

Figure 14-5. The **-links** Option with **find**

AU1310.0

Notes:

Using the **-links** option

-links +1 will list names of files that have more than one link associated with them. It is necessary to use **-type f** to narrow the search to files only, since directories, by nature, all have at least two links.

Instructor Notes:

Purpose — Describe the use of the `-links` with the `find` command.

Details — Using `-links` is probably a good security exercise for you to perform periodically to ensure no one has linked to one of your programs without your permission. `ls -li` will show you the number of links and the i-node numbers so you know exactly which files are linked to each other. Do not forget to perform the `find` on only ordinary files, as directories all have at least two links associated with them (the entry in the parent directory and its own).

Additional Information — Rather than using `xargs` with the `find` in the `-links +1` example, a pure `find` could have been used:

```
find . -type f -links +1 -exec ls -li {} \;
```

Transition Statement — Let's wrap up the discussion on `find` by looking at using `alias` to cut down on rekeying this command every time you want to use it.

alias and find

```
$ cat $HOME/.kshrc          ENV=$HOME/.kshrc
alias mylinks='find . -type f -links +1 | xargs ls -li'
alias myrm='find . -type f -mtime +30 | xargs rm'

$ mylinks
127 -rw-r--r-- 3 team01 staff ... /home/team01/myfile
127 -rw-r--r-- 3 team01 staff ... /home/team01/yourfile
127 -rw-r--r-- 3 team01 staff ... /home/team02/akafile

$ myrm
```

© Copyright IBM Corporation 2008

Figure 14-6. alias and find

AU1310.0

Notes:

Simplifying long commands

Aliases can be used to simplify a very long command that you may need to run on a periodic basis. In the examples shown in the visual, you can issue just the single alias rather than a lengthy command.

Aliases are a handy mechanism to cut down on the keystrokes used to enter a command and its parameters. As you learned earlier in this course, the best way to define an alias is to put the definition into the **.kshrc** file.

Instructor Notes:

Purpose — Describe using `alias` to allow a user to issue another command in place of the original command.

Details — Point out that setting the alias from the command line will only be effective for this login session. From previous units, students should already know how to set up aliases using the `ENV` variable and the `.kshrc` file to ensure that they are available upon each login and passed to each child process as needed.

Additional Information —

Transition Statement — Now, let's look at some commands that will show you where a command is located.

which, whereis, and whence

```
$ which find grep  
/usr/bin/find  
/usr/bin/grep  
  
$ whereis find grep  
find: /usr/bin/find  
grep: /usr/bin/grep  
  
$ whence -pv find grep  
grep is /usr/bin/grep  
find is /usr/bin/find
```

© Copyright IBM Corporation 2008

Figure 14-7. **which**, **whereis**, and **whence**

AU1310.0

Notes:

Introduction

Once you know the exact name of the command you need to use, you may discover that you need to know the full path name of where that program resides. This can happen if the directory that contains the command is not in your `PATH` or if you are writing a program that requires the full path to a command.

Finding command locations

From what we have learned so far, you could use `find`. It would look through the entire tree structure and match full path names for the command. Any one of these three commands can find the full path name, and they all are easier than keying in the syntax for `find`.

The `which` command takes a list of program names and looks for the files that run when these names are given as commands. It will only show the first instance of the

command that you listed. If you are using the C shell and have a **.cshrc** file it will also check for aliases.

The **whereis** command attempts to find the desired program from a list of standard locations. Also, **whereis** does not search your shell's search PATH so it may not find shell scripts in local system directories or in your **bin** directory. If the argument is located in multiple locations, it will list them all. For example, try using the argument **passwd** with both commands.

whence is a built-in command specific to the Korn shell. It is very similar to the **which** command, except that it will also check for KSH aliases.

Instructor Notes:

Purpose — Discuss the purpose of these three commands.

Details — A common task is to examine the contents of a shell script. However, you may not know where the executable file is located, and may not even be sure that it is a script or an executable program. In this situation, you can use the **whence** command in combination with command substitution.

For example, to determine what type of file we are dealing with:

```
$ file `whence <command-name>'
```

And then to examine the file:

```
$ more `whence <command-name>'
```

Transition Statement — Now that you know how to find where a file is located, let's look at a command that will show you the type of file data that is in a file.

file

```
$ file /usr/bin/vi
/usr/bin/vi:executable (RISC System/6000) or object module

$ file c1
c1:      ascii text

$ file /usr/bin
/usr/bin:  directory

$ ls > filenames
$ cat filenames
c1
dir1
$ file -f filenames
c1:      ascii text
dir1:    directory
```

© Copyright IBM Corporation 2008

Figure 14-8. `file`

AU1310.0

Notes:

Introduction

To find out whether it makes sense to display a file on the terminal, use the `file` command to determine the type of data in the file. It reads each file and tries to decide whether it is simple ASCII text, a directory, c program, and so forth.

The `file` command

This can be useful for a couple of reasons. First, it can tell you what files are readable before you potentially hang your terminal by trying to display an executable file. Second, it can help you determine what kind of a binary file it is and what operating system version it was compiled under.

Command details

The file command uses the `/etc/magic` file to identify what kind of file is involved by looking in the contents for a *magic cookie*. A magic cookie is a numeric or string constant that is indicative of a particular type of file. The `/etc/magic` file is the list of magic cookies and the kind of file each is related to.

Using `file` on a non-existent file results in an error message stating that it could not get a file status.

When using `file` with the `-f` option against a list of file names within a single file, each file name must appear alone on a line.

Instructor Notes:

Purpose — Describe the use of the `file` command to determine the type of a file.

Details — The `file` command uses a magic file to identify various file types. If a code is present, it knows that this file is the corresponding type of file. You can look in the `/etc/magic` file that is used by the `file` command.

Be aware that the magic cookie method is not perfect and the `file` command can sometimes give misleading results if a file has a string which is related to a different kind of file.

Additional Information — You can also string out a list of file names such as `file /usr/bin/vi c1 /usr/bin` and get one line of information for each file listed.

Transition Statement — This unit is introducing so much information, we have split up the exercises into two files.

Exercise: AIX Utilities (3)



© Copyright IBM Corporation 2008

Figure 14-9. Exercise: AIX Utilities (3)

AU1310.0

Notes:

After completing the exercise, you will be able to:

- Use the `find`, `xargs`, and `file` commands

Instructor Notes:

Purpose — To introduce the next lab.

Details — Describe the goals of the next lab.

Additional Information —

Transition Statement — Let's continue with some more utilities starting with the `diff` command.

diff (Differential File Comparator)

- Analyzes text files
- Reports the **differences** between files

```
diff [-options] file1 file2
```

© Copyright IBM Corporation 2008

Figure 14-10. **diff** (Differential File Comparator)

AU1310.0

Notes:

Comparing two files

There may be times when it is useful to find the differences between two files. **diff** will display the lines that are different and reports them in such a way that you can automatically create a script to make them identical or to change just certain lines so they match in both files. It can also compare all the text files in two directories.

diff only works with files that are text files. The output of **diff** tells you which lines must be changed in the first file to make both files agree.

Instructor Notes:

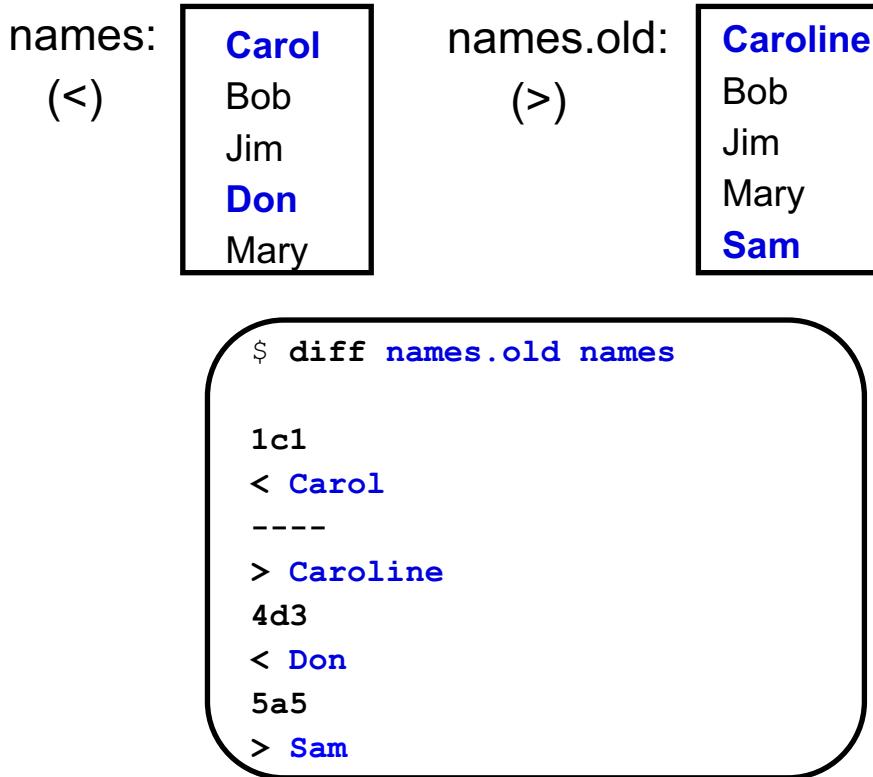
Purpose — Describe the features of `diff`.

Details —

Additional Information —

Transition Statement — Let's take a look at an example.

Comparing Two Files Using `diff`



© Copyright IBM Corporation 2008

Figure 14-11. Comparing Two Files Using `diff`

AU1310.0

Notes:

Example of `diff` output

When looking at `diff` output, lines starting with < show lines which are in the first file, but do not appear in the second file. Lines starting with > show new lines that appear in the second file but are not in the first file. Lines changed between the two files show as both < and >. In the output, there are `ed` line editor subcommands that will convert the first file to the second. These codes are:

- a Indicates lines should be added or appended to the first file in order to obtain the result shown in the second file
- d Indicates lines have been deleted from the second file
- c Indicates lines that have been changed between the first file and the second file

diff command options

Use the **-w** option to tell **diff** to ignore spaces and tabs.

Use the **-b** option to tell **diff** to ignore leading spaces and tab characters and consider all other strings of spaces as equal.

diff -e produces output in a form suitable for use with the **ed** line editor to convert the first file to match the second file.

Instructor Notes:

Purpose — Look at an example of `diff` output.

Details — When working on a project, it is not uncommon to have a couple of versions of a document in your directory. One may be the original and the other one that you have edited recently. It is an easy way to examine changes in a document without reading the entire document, or you may have different versions of programs such as a recent phone directory and old phone directory.

The output in the example shows three changes or differences between `names` and `names.old`.

Line 1 in `names` is Carol; whereas, line1 in `names.old` is Caroline. The output is using both < and > thus indicating that if you want `names` to be like `names.old`, Carol has to be changed to Caroline.

The next difference is on line 4 of `names` and 3 of `names.old`. This is indicating that if you delete line 4 in the first file, the two files will be in sync from there on. Notice that just the < is indicated meaning this line in the first file does not appear in the second file.

The last difference indicates that an additional line has been added to the second file. If you want the two files to agree, this line would have to be added or appended after line 5 in the first file. The new line is still line 5 in the second file because the first file had an additional line that does not appear in the second file.

Additional Information — The order of the specified file names is important. If you reversed the two file names in this example, the results would be:

```
1c1
< Caroline
---
> Carol
3a4
> Don
5d5
< Sam
```

There is a `diff3` that can be used to compare 3 files. Differences are indicated by === if all 3 files differ; ===1 if only the first file differs; ===2 if only the second file differs; ===3 if only the third file differs. The output is also preceded by the name of the file to help keep the output clear.

There is also an `sdiff` that does a side-by-side comparison of files which may be easier to interpret. It uses a < to point to lines that are only in the first file; > for lines only in the second file; and | for lines that are in both, but different. By default this command shows all the lines in both files.

Transition Statement — Let's take a look at a command that can be used on any type of file, not just text files.

Comparing Two Files Using `cmp`

```
$ cmp names names.old
names names.old differ: byte 6, line 1

$ cmp -l names names.old
6    12   151
7   102   156
8   157   145
...
...
...
cmp: EOF on names
```

© Copyright IBM Corporation 2008

Figure 14-12. Comparing Two Files Using `cmp`

AU1310.0

Notes:

Comparing files byte by byte

Unlike `diff`, which only compares text files, `cmp` can compare all types of files. It will read two files until it finds the first difference and then reports exactly which byte is different.

In the first example, the first byte that was detected as different between the two files is byte 6 on the first line.

For a more detailed comparison, the `-l` option will list all the bytes that are different. The first column is the decimal value of the byte number, the second column is the octal value of the byte in the first file, and the third value is the octal value of the byte in the second file.

In the second example, the sixth byte in `names` is octal 12, and in `names.old` the octal is 151. For text files, the octal values are the characters as they are represented by the ASCII character set.

Instructor Notes:

Purpose — Discuss how `cmp` differs from `diff`.

Details — `diff` compares text files only. It also uses the concept of lines to report the differences between two files and it prints the contents of the files to the screen. While `diff` looks for lines that are different, `cmp` does a byte-by-byte comparison. On text files it does a character-by-character comparison.

The first column prints the decimal value of the byte, the octal value of the byte in the first file, and the octal value of the byte in the second file. For text files, as shown here, the octal values are values of the characters as they are represented by the ASCII character set.

We are using the same two files that were used with the `diff` example. The output of the first example shows that the name Carol in `names` and Caroline in `names.old` is where the first difference occurs...which is byte 6.

As you can see, it is not very useful for showing the differences between text files. It is more suited for program object and data files; however, it does provide a quick way to find out whether files are different or not. If the files are different you can use `diff` to get details of the differences.

Additional Information — The `comm` command can tell you what information is common to two files and what information appears uniquely in one file or the other. By default, it uses a 3-column output consisting of 1) lines that are only in the first file; 2) lines that are only in the second file; 3) lines that are in both files.

Transition Statement — Now that we have covered how to compare the contents between files, let's look at a command that will compare the contents between two directories.

Comparing Directories Using `dircmp`

```
$ dircmp -d /home/team01 /home/team02
```

```
Fri Jan 21 10:31:10 CDT 2000 /home/team01 only and /home/team02 only
./dir1
./dir1/c3
./dir1/c4
./dir1/dir2
./dir1/dir2/c5
./dir1/dir2/c6
```

1: List files **unique to each directory**

```
Fri Jan 21 10:31:10 CDT 2000 Comparison of /home/team01 and /home/team02
directory      .
same          ./profile
different     ./sh_history
different     ./c1
same          ./c2
```

2: List files with **identical names**

```
Fri Jan 21 10:31:10 CDT 2000 diff of ./c1 in /home/team01 and /home/team02
1c1
< Now  is the time for all good men
---
> Now  is the time for all good women
```

3: Display **differences for common files**

© Copyright IBM Corporation 2008

Figure 14-13. Comparing Directories Using `dircmp`

AU1310.0

Notes:

Comparing directories

The `dircmp` command compares the two directories specified and writes information about their contents to the display.

First, it lists the files unique to each directory.

Second, it lists the files with identical names in both directories and lets you know if the contents are the same or different.

Third, it displays for each common file name both versions of the differing file contents. The display format is the same as that for the `diff` command.

Command options

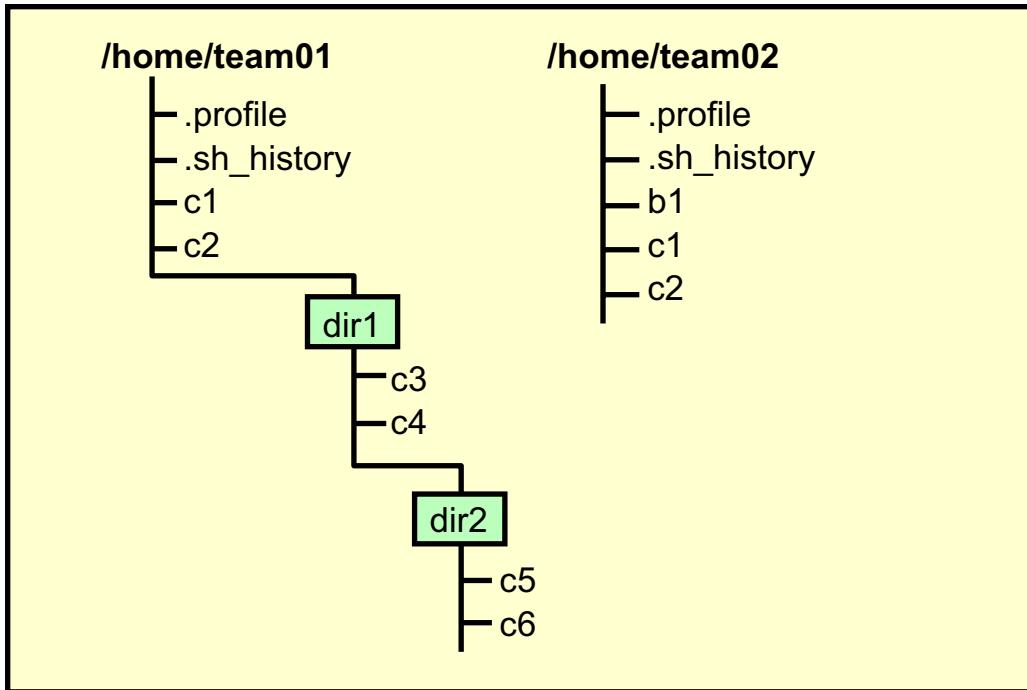
The `-d` option lists the `diff` output which is displayed last. The `-s` option could have been listed to silence or not display the files that are named the same and have identical contents, as indicated by the word `same` in the second area of information.

Command output

Be sure to pipe the output of the `diramp` command to `pg` or `more` as it will produce multiple pages of output.

Directory structure used in the examples

Sample Directory Structure



© Copyright IBM Corporation 2008

Instructor Notes:

Purpose — Describe how to compare the contents of two directories.

Details — Cover the information in the student notes using the tree structure to help explain the output. Direct them to `diff -r`, which displays output similar to the third area of information.

If just the `dircmp` command was used with no options, the output would have produced just the first two sections of output.

Additional Information — Following is the output if `diff -r dir1 dir1/dir2` were executed for `team01`:

```
Only in dir1: c3
Only in dir1: c4
Only in dir1/dir2: c5
Only in dir1/dir2: c6
Only in dir1: dir2
```

The `dircmp` gives you more detailed information.

Transition Statement — Let's now focus on how to compress files to save disk space.

compress, uncompress, and zcat

```
$ ls -l file1
-rw-r--r-- 1 team01 staff 13383 July 26 10:10 file1
$ compress -v file1
file1: Compression 56.99% file1 is replaced with
file1.Z
$ ls -l file1.Z
-rw-r--r-- 1 team01 staff 5756 July 26 10:10 file1.Z
$ zcat file1.Z
(output is the normal output of the uncompressed file)

$ uncompress file1.Z
$ ls -l file1
-rw-r--r-- 1 team01 staff 13383 July 26 10:10 file1
```

© Copyright IBM Corporation 2008

Figure 14-14. compress, uncompress, and zcat

AU1310.0

Notes:

Introduction

To save disk space when files are saved, use the **compress** command. The file is compressed without deleting the information it contains. This is extremely useful if you are frequently exchanging files either over the network or via diskette or tape.

Compressing files

The **compress** command compresses data, using Lempel-Zev coding to reduce the size of files. Each file is replaced by a compressed file with a **.Z** appended to its name. The compressed file retains the same ownership, modes, and modification time of the original file. The **-v** option writes the percentage of compression that took place.

The **compress** command will delete the file it is compressing and replace it with the compressed file renaming it with a **.Z** extension.

If compression does not reduce the size of a file, a message is written to STDERR and the original file is not replaced.

Viewing compressed files

There is no need to uncompress the file to read it. The `zcat` command allows the user to expand and view a compressed file without uncompressing that file. It does not rename the expanded file or remove the `.Z` extension. It simply writes the expanded output to STDOUT.

Uncompressing files

The `uncompress` command restores the original file that was compressed by the `compress` command. Each compressed file is removed and replaced by the expanded copy. The expanded file has the same name as the compressed version without the `.Z` extension.

Instructor Notes:

Purpose — Cover the output from all three commands.

Details — The amount of compression depends on the size of the file, the number of bits per code specified by the Bits variable, and the distribution of common substrings. Typically source code or text is reduced by 50 to 60%.

Additional Information — Usually the `compress` command is more compact and takes less time to compute than the compression achieved by the Huffman coding used in the `pack` command.

The `pack` command will replace the current files with a compressed file using the original name appended with a `.z` (lowercase z). `uncompress` and `zcat` can not restore a packed file. You would need to use the `unpack` and `pcat` commands with such files. (The lowercase `z` is part of a checkpoint question).

Transition Statement — When looking at a file, how can you tell if tabs or spacing were used? What about other non-printing characters? Let's take a look to see how we can view those non-printable characters.

Displaying Non-Printable Characters in Files

```
$ cat myfile
```

This file has tabs and spaces and ends with a return

```
$ cat -vte myfile
```

This^Ifile^G has tabs^Iand spaces and^Iends with a^Ireturn\$

-v: Display non-printing characters as visible characters

-t: Display tab characters as ^I

-e: Display a \$ at the end of each line

© Copyright IBM Corporation 2008

Figure 14-15. Displaying Non-Printable Characters in Files

AU1310.0

Notes:

Finding non-printable characters

There will be times when you will need to know if tabs or spaces were used in a file or what is causing the file to appear different when using `diff`, but you cannot see anything visibly different.

Using the `cat` command with these three options will give you a good idea of how the file was created:

`-v` displays non-printing characters as visible characters.

`-t` displays tab characters as ^I.

`-e` displays a \$ at the end of each line.

Instructor Notes:

Purpose — Describe the use of the three options that can be used with the `cat` command to display non-printable characters.

Details — The `^I` indicates tabs so you know that some spacing was used between some of the words. The `^G (Ctrl+G)` must have been accidentally keyed as it is showing up in the middle of the words file and has. You can see that the line was terminated with the use of a return with the presence of a `$` (dollar sign).

Additional Information —

Transition Statement — This is great for examining the contents of a file, but what about the contents of a directory?

Non-Printable Characters in Directories

```
$ ls
greatfile myfile
$ rm greatfile
No such file
$ ls | cat -vt
^Ggreatfile
myfile
```

To fix this file, use one of these three methods!

1. rm ^Ggreatfile
2. mv ^Ggreatfile greatfile
3. ls -i
130 ^Ggreatfile 127 myfile
find . -inum 130 -exec rm {} \;

© Copyright IBM Corporation 2008

Figure 14-16. Non-Printable Characters in Directories

AU1310.0

Notes:

Finding non-printable characters in directories

There are times when you list the contents of a directory and you see the file you want to work with, but you cannot access it. It may be that you accidentally pressed a control character while creating the name of the file.

View the contents of the directory by piping the output of **ls** to **cat** using its varied options. Identify what the problem is with the file name you are trying to access.

There are three methods of fixing the file name:

- a. If you do not need the file any longer, remove it but ensure you key in the control character as part of the name.
- b. If you need to keep the file, rename it, also ensuring that you include the control character as part of the source file name.
- c. If you cannot remove the file using method 1, find the i-node number of the file and use the **find** command with the **-inum** expression.

Instructor Notes:

Purpose — Describe how to view non-printable characters in directory contents.

Details — Cover the three methods listed on the visual and in the student notes.

Additional Information —

Transition Statement — How can you ensure that an application or you assign unique file names?

Assigning Unique File Names

```

$ touch myfile$$ ← Append the process ID: $$

$ ls
myfile1288

$ date
Mon Feb 14  07:20:15      CDT 2001
$ date +'%-m%-d%H%M%S'
0214072015
$ touch myfile.$(date +'%-m%-d')
$ ls
myfile.0214

```

Append the date using a command substitution

© Copyright IBM Corporation 2008

Figure 14-17. Assigning Unique File Names

AU1310.0

Notes:

Introduction

If you have to ensure that an application or you always assign a unique file name when you create a file, use one of these two methods to automatically make that happen.

Appending process ID

The shell will automatically append the process ID to the file name when the \$\$ parameter is used. This will append a process ID from two to five characters.

Appending the date

The output of the **date** command is built by using the format described by the % variables. The + parameter allows you to change the output format. This example shows taking just the month and date and appending it as an extension to the filename.

Instructor Notes:

Purpose — Describe two methods that can assign unique file names.

Details — You may want to use some of the additional fields from the output of the `date` command to ensure uniqueness. If two of you are naming files `myfile` on the same day, using the month and day will not necessarily ensure unique names.

Additional Information —

Transition Statement — I would like to cover a few checkpoint questions before we go to the lab.

Checkpoint

1. True or false? **find**'s most important characteristic is its ability to travel up through the file tree hierarchy.
2. True or false? When quoted metacharacters are used with **find**, the shell will first expand the wildcard then pass control to find.
3. Which command is used to determine the type of data in a file?
`cmp`
`diff`
`file`
`dirncmp`
4. True or false? **diff** compares text files only.
5. True or false? The **compress** command will delete the file it is compressing and replace it with the compressed file also renaming it with a `.z` extension.
6. To display non-printable characters in a file or directory, use:
`ls -li`
`cat -vte`
`diff -c`
`cmp`

© Copyright IBM Corporation 2008

Figure 14-18. Checkpoint

AU1310.0

Notes:

Instructor Notes:

Purpose — To test the students' understanding of the material presented.

Details —

Checkpoint Solutions

1. True or false? `find`'s most important characteristic is its ability to travel up through the file tree hierarchy. **False**.
2. True or false? When quoted metacharacters are used with `find`, the shell will first expand the wildcard then pass control to `find`. **False**.
3. Which command is used to determine the type of data in a file?
`cmp`
`diff`
file
`aircmp`
4. True or false? `diff` compares text files only. **True**.
5. True or false? The `compress` command will delete the file it is compressing and replace it with the compressed file also renaming it with a `.z` extension. **False. The extension is an uppercase Z**
6. To display non-printable characters in a file or directory, use:
`ls -li`
cat -vte
`diff -c`
`cmp`

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — Let's test what you have learned in a lab exercise.

Exercise: AIX Utilities (4)



© Copyright IBM Corporation 2008

Figure 14-19. Exercise: AIX Utilities (4)

AU1310.0

Notes:

After completing the exercise, you will be able to:

- Use `diff`, `cmp`, and `dirmp` to compare files and directories
- Use `compress`, `zcat`, and `uncompress`
- Use `cat` to display non-printable characters

Instructor Notes:

Purpose — To introduce the next exercise to the students.

Details — Describe what the students will learn during this exercise.

Additional Information —

Transition Statement — Before we move on to a new unit, let me summarize the key points from this one.

Unit Summary

- **xargs** reads arguments one line at a time from S/I and assembles as many of them as will fit into one command line.
- **-links** searches for the **number of links** in files or directories.
- **which**, **whereis** and **whence** are used to **locate programs**.
- **diff** compares the contents of two **text files**.
- **cmp** compares the contents of two files of **all file types**.
- **dir cmp** is used to **compare** the contents of two **directories**.
- **compress** compresses data in files using Lempel-Zev coding.
- **cat -vte** displays non-printable characters in a file or directory.

© Copyright IBM Corporation 2008

Figure 14-20. Unit Summary

AU1310.0

Notes:

Instructor Notes:

Purpose — Summarize this unit's key points.

Details —

Additional Information —

Transition Statement — Let's move on to a new unit.

Unit 15. Additional Shell Features

Estimated Time

00:45

What This Unit Is About

This unit introduces basic shell programming concepts.

What You Should Be Able to Do

After completing this unit, students should be able to:

- Pass positional parameters to shell scripts
- Use the `test` command
- Use the `if` statement
- Implement interactive shell scripts
- Implement loops within scripts

How You Will Check Your Progress

Accountability:

- Student activity
- Checkpoint questions
- Exercise

Unit Objectives

After completing this unit, you should be able to:

- Pass **positional parameters** to shell scripts
- Use the **test** command
- Use the **if** statement
- Implement **interactive shell scripts**
- Implement **loops** within scripts

© Copyright IBM Corporation 2008

Figure 15-1. Unit Objectives

AU1310.0

Notes:

Instructor Notes:

Purpose — To introduce the objectives for this unit to the students.

Details — Explain that this unit is just a light exposure to shell script writing, and that there is a full course on Korn Shell Programming available.

Additional Information —

Transition Statement — Let's start this unit with a look at shell variables.

Important Shell Variables

\$\$	Process ID (PID)
\$0	Shell script name
\$#	Number of arguments passed to the shell script
\$*	All command line arguments passed to the script
\$?	Exit value of the last command
\$!	Process ID of last background process

© Copyright IBM Corporation 2008

Figure 15-2. Important Shell Variables

AU1310.0

Notes:

Shell variables

These variables are set by the shell or a shell script and can, therefore, be referenced by the user or shell script:

- \$\$ Contains the process ID of the current executing process.
- \$0 Contains the name of the shell script that is currently executing.
- \$# Is the number of positional parameters passed to the shell, not counting the name of the shell procedure itself.
- \$* Contains the value of all positional parameters passed to the shell, not including the name of the shell procedure itself.
- \$? Is the exit value of the last command executed. Its value is a decimal string. For most commands 0 indicates a successful completion.
- \$! Is the process number of the last process run in the background.

Instructor Notes:

Purpose — Define these important special variables.

Details — Explain each variable in turn, mentioning how the value of the variables can be extracted with the `echo` command. For example, to find out the PID of the executing process enter `echo $$`.

Additional Information —

Transition Statement — Let's look at a few other special variables.

Positional Parameters

Parameters can be passed to shell scripts as **arguments** on the command line:

```
$1, $2, ... $9  
${10}, ${11}, ... ${n}      (Korn Shell only)
```

```
$ cat para_script
echo First Parameter entered was $1
echo Second Parameter entered was $2
echo Third Parameter entered was $3

$0      $1  $2  $3
$ para_script Good Day Sydney
First Parameter entered was Good
Second Parameter entered was Day
Third Parameter entered was Sydney
```

© Copyright IBM Corporation 2008

Figure 15-3. Positional Parameters

AU1310.0

Notes:

Using positional parameters

Parameters can be passed to shell scripts as arguments on the command line. They are implemented in the script by `$n` where `n` is the position on the command line after the command.

In the Bourne shell you cannot reference more than nine arguments at once.

Instructor Notes:

Purpose — Discuss how positional parameters can be used to pass argument values to a shell script.

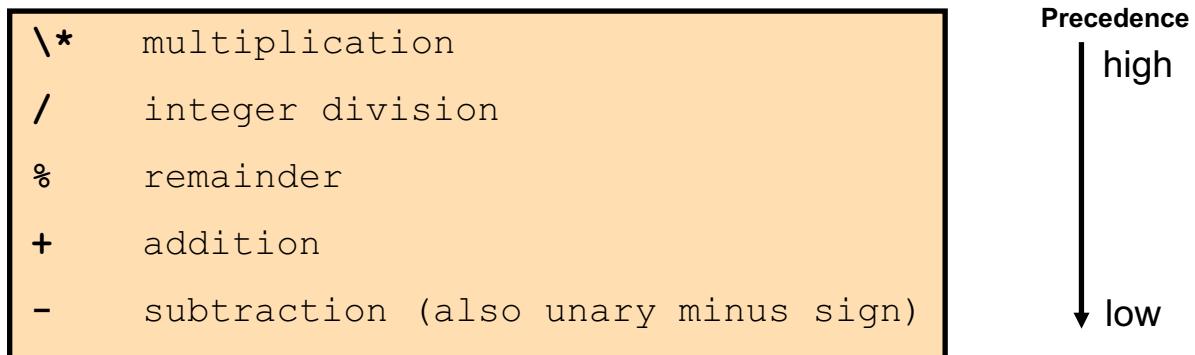
Details — Positional parameters enable users to create generic shell scripts into which they can pass different values. The method described is the Korn shell syntax. By using the curly braces {} and a number, the user can address as many parameters on the command line as required. Without the curly braces, the Korn shell can use a single number (0-9) to identify variables to reference the first 10 positional variables. In the example shown, \$1 refers to the first argument the string `Good`, \$2 refers to the second argument the string `Day`, and \$3 to the third argument `Sydney`. This way these values are passed back to the script.

Additional Information — The Bourne shell uses a slightly different notation. Using the Bourne shell you can only reference variables \$1 to \$9. To reference past the 9th variable the `shift` command must be used. This is to be used as information for the instructor, and should only be explained if there is a question regarding the Bourne shell syntax.

Transition Statement — Let's see how we can perform arithmetic expressions using the `expr` command.

The `expr` Utility

- Use the `expr` utility to perform [integer arithmetic](#).
- `expr` offers the following **operators**:



© Copyright IBM Corporation 2008

Figure 15-4. The `expr` Utility

AU1310.0

Notes:

Introduction

The `expr` command reads the expression parameter, evaluates it, and writes the result to standard output.

The operators are shown here in order of precedence: highest to lowest.

`expr` expression parameter

You must apply the following rules to the expression parameter:

- Spaces are required between operators and expressions except for the unary minus with a literal value, such as -3.
- Precede special characters to the shell with a \ (backslash). For example, * is used to express multiplication.
- Quote strings containing blanks or other special characters.

Exceptions

`expr` only handles integer arithmetic. It cannot handle values that are non-integer and will not attempt any calculations based on non-integer values.

For example, `expr 3.5 + 5.7` gives the error message:

```
expr: 0402-046 A specified operator requires numeric parameters
```

Precedence

Precedence refers to the order in which a mixture of arithmetic operations are executed. If I write `6+4/2`, default precedence states that the division will be done first, giving an answer of 8. If I wish to overrule default precedence, I would use parentheses such as `(6+4)/2`. Now the addition must be done before I divide, resulting in an answer of 5.

Instructor Notes:

Purpose — Illustrate the `expr` command.

Details —

Additional Information —

Transition Statement — Let's look at some examples.

expr Examples

```
$ var1=6
$ var2=3
$ expr $var1 / $var2
2
```

```
$ expr $var1 - $var2
3
```

=> Use `\(\)` to group expressions:

```
$ expr \( $var1 + $var2 \) \* 5
45
```

=> Use `command substitution` to store the result in a variable:

```
$ var3=$(expr $var1 / $var2)
$ echo $var3
2
```

© Copyright IBM Corporation 2008

Figure 15-5. `expr` Examples

AU1310.0

Notes:

Command examples

The visual shows some `expr` commands.

You must group expressions, if you do not want to use the default precedence.

If you want to store the result of the `expr` command in a variable, you must use command substitution.

Instructor Notes:

Purpose — Discuss the practical uses of the `expr` command.

Details — Go through each example, and point out that the normal mathematical rules of precedence apply.

Emphasize the need to escape the shell metacharacter meaning of the parentheses when used for precedence control. Contrast that with the use of parentheses in the command substitution example.

Additional Information —

Transition Statement — Let's see how conditional execution can be specified.

Conditional Execution

The **exit value** from a command or group of commands can be used to determine **whether to do the next command**:

command1 && command2

if (command1 successful) then do (command2)

\$ ls s* && rm s*

command1 || command2

if (command1 not successful) then do (command2)

\$ cd /dir1 || echo Cannot change to /dir1

© Copyright IBM Corporation 2008

Figure 15-6. Conditional Execution

AU1310.0

Notes:

Conditional examples

In the visual are more examples of a list of commands or command grouping, similar to the unconditional list we covered earlier using a semicolon as the separator. The difference is that these sequences are *conditional*.

In the first example, with the **&&** symbol, if the first command is successful, then the second command will be executed. For example, if there are any files that begin with **s**, they will be removed.

In the second example, the **||** symbol causes the command following it to be executed only if the preceding pipeline returns a non-zero exit value. Either the **cd** command will execute successfully, or an error message will be given.

Instructor Notes:

Purpose — Define the use of conditional execution.

Details —

Additional Information —

Transition Statement — Let's look at another useful AIX command: the `test` command.

test Command

The **test** command allows you to test for a given condition:

```
test expression or [ expression ] or [[ expression ]]
```

The **test** command evaluates the expression and returns **true** or **false**

Operator:	Returns true, if:
<code>\$string1 = \$string2</code>	Strings are identical
<code>\$string1 != \$string2</code>	Strings are not identical
<code>\$number1 -eq \$number2</code>	Numbers are equal
<code>\$number1 -ne \$number2</code>	Numbers are not equal
<code>-e \$file</code>	File exists
<code>-d \$file</code>	File is a directory
<code>-r \$file</code>	File is readable
<code>-w \$file</code>	File is writable

© Copyright IBM Corporation 2008

Figure 15-7. **test** Command

AU1310.0

Notes:

Using the **test** command

The **test** command can be used to evaluate an expression and returns an exit code of 0 if it is true. It has a number of different formats. If the square braces [...] are used, then spaces must be left between each brace and the expression which is specified.

In newer Korn shell scripts the modern notation [[...]] is used very often, which is an extension of the **test** command.

Instructor Notes:

Purpose — Define the syntax of the `test` statement.

Details — This is a very useful command, and can be used to test for particular conditions. In this course only a few of the conditions which can be tested for are listed.

Additional Information —

Transition Statement — The `test` command is usually used in conjunction with the `if` statement. Let's see how these two commands work together.

if Command

```
if condition is true
then
    carry out this set of actions
else
    carry out these alternative actions
fi
```

} optional

```
$ cat active
USAGE="$0 userid"

if [[ $# -ne 1 ]]
then
    echo "Proper Usage: $USAGE"
    exit 1
fi

if who | grep $1 > /dev/null
then
    echo "$1 is active"
else
    echo "$1 is not active"
fi

exit
```

```
$ cat check_user
USAGE="$0 username"

if [[ $# -ne 1 ]]
then
    echo "Proper usage: $USAGE"
    exit 2
fi

grep $1 /etc/passwd >/dev/null
if [[ $? -eq 0 ]]
then
    echo "$1 is a valid user"
    exit 0
else
    echo "$1 is not a valid user"
    exit 1
fi
```

© Copyright IBM Corporation 2008

Figure 15-8. if Command

AU1310.0

Notes:

Introduction

The **if** statement can be used to control the flow of the program and the commands to be executed.

Controlling the flow of a script

The first line (**if** statement) evaluates the return value (true or false) of the command (such as **test**) following the **if** keyword. Depending on the result, if the **if** statement evaluates to true, then the commands after the **then** statement are executed. If, however, the **if** statement evaluates to false, then, the commands after the **else** statement are executed.

You do not always need an **else** statement, but if you include it there can be only one within an **if** clause.

As soon as a true expression is found, the corresponding block of commands is executed. Then the flow of the program will continue after the closing `fi` statement.

Exiting a script

The `exit` statement is used to terminate a process. If the shell script executes successfully, a value of `0` is returned. An exit code that is not equal to `0`, indicates an error. The `exit` statement allows you to control the exit code when terminating your script by following the `exit` command with a numeric value.

Use the `$?` variable in the shell to display the exit value of the prior command, including shell scripts.

Additional information

The visual shows two shell scripts, `active` and `check_user`, that will be used in the an activity that we will do shortly.

Instructor Notes:

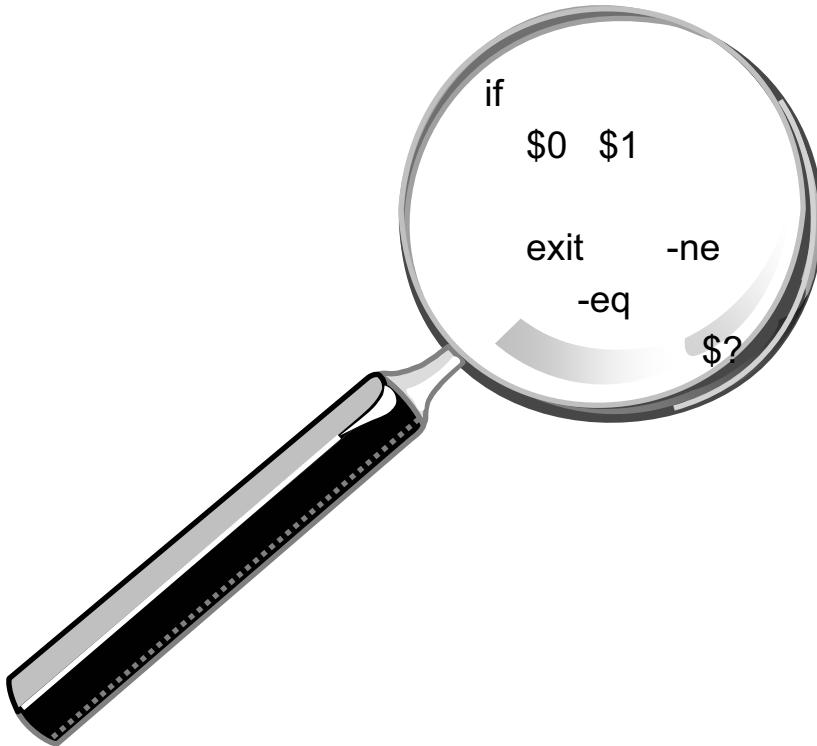
Purpose — Illustrate conditional execution using `if`.

Details — Explain one of the two examples. These shell scripts will be covered in the next activity. Do not explain the `check_user` shell script, since that is for the students to analyze and explain in the activity.

Additional Information —

Transition Statement — Let's have an activity.

Activity: Writing Shell Scripts



© Copyright IBM Corporation 2008

Figure 15-9. Activity: Writing Shell Scripts

AU1310.0

Notes:

- ___ 1. Log in to your system.
- ___ 2. The last visual shows a shell script check_user. Create and execute this shell script. Do not forget to define the script as an *executable* file.
- ___ 3. Analyze the script and try to figure out how it works. Answer the following questions:
 - a. What is \$# ?
 - b. What is \$? ?

-
- c. Look at the following two lines:

```
grep $1 /etc/passwd >/dev/null  
if [[ $? -eq 0 ]]
```

How can you write these two lines in one line? Tip: Compare the scripts `check_user` and `active`.

Instructor Notes:

Purpose — To give the students a chance to test the information presented so far.

Details — Give the students some time to review what they learned. Also review the questions from the activity:

1. What is \$# ?

Answer: Number of passed command line arguments.

2. What is \$? ?

Answer: Exit code of `grep`. If 0, `grep` found the corresponding user.

3. Look at the following two lines:

```
grep $1 /etc/passwd >/dev/null  
if [[ $? -eq 0 ]]
```

How can you write these two lines in one line?

```
if grep $1 /etc/passwd >/dev/null
```

Additional Information —

Transition Statement — Introduce the `read` command.

read Command

The **read** command reads one line from standard input and assigns the values of each field to a shell variable.

```
$ cat delfile

# Usage: delfile
echo "Please enter the file name:"
read name
if [[ -f $name ]]
then
    rm $name
else
    echo "Error: $name is not an ordinary file"
fi
```

© Copyright IBM Corporation 2008

Figure 15-10. **read** Command

AU1310.0

Notes:

Using the **read** command

The **read** command can be used to assign more than one variable value. If more than one argument is read into the script, the first argument would be assigned to the first variable name defined by the **read** statement, the second argument to the second variable name, and so on until the last field is reached.

If there are more arguments supplied than variable names defined, then the last variable name is given the value of all the remaining fields.

The example does not show testing for the file permissions. This would also have to be in effect.

Note: The # indicates a comment in a shell script. Everything right to the # is not interpreted by the shell.

Instructor Notes:

Purpose — Demonstrate how interaction can be achieved within a script.

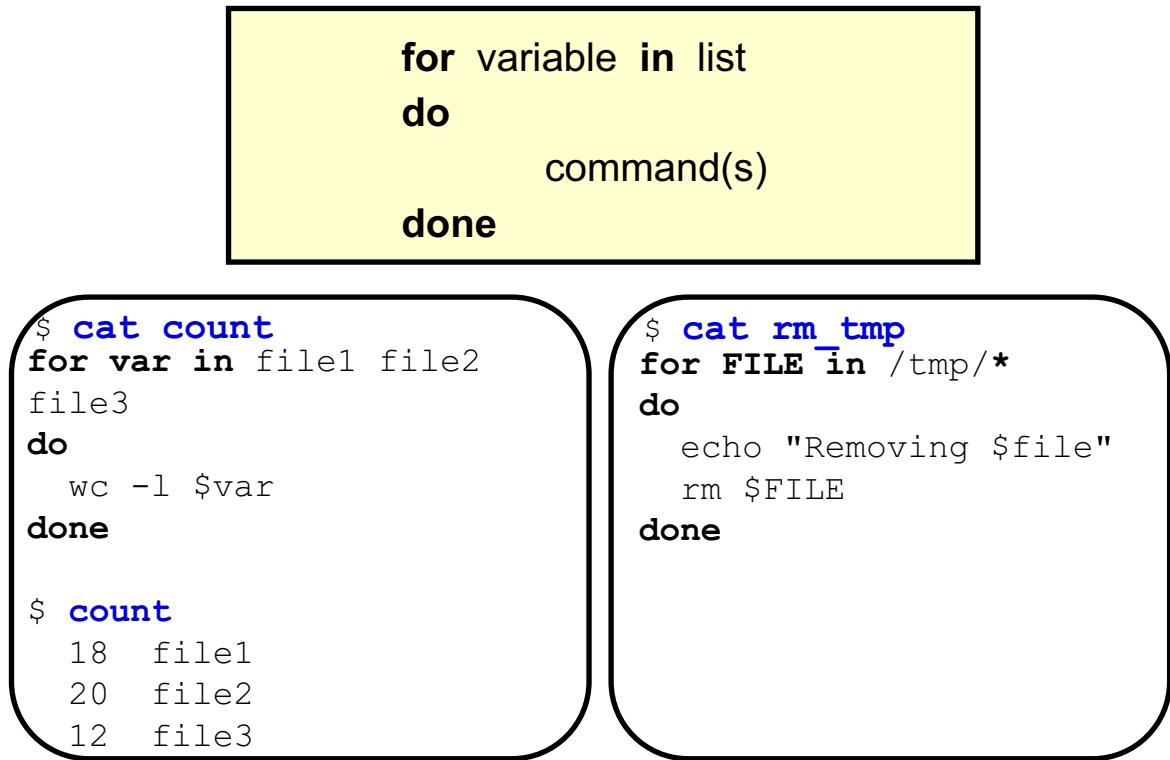
Details — The `read` command allows for breaks in the flow of a shell script where the user is asked for a piece of information. An action is then carried out based on the input value.

The example given is a simple script which asks the user to input a file name which the user wishes to delete. The user's value is taken and tested to see if the file specified is an ordinary file. If it is, then it is deleted, and if it is not, then an error message is given.

Additional Information —

Transition Statement — Let's see how we can implement looping statements in shell scripts.

for Loop Syntax



© Copyright IBM Corporation 2008

Figure 15-11. for Loop Syntax

AU1310.0

Notes:

Introduction

The **for** loop allows you to repeat a section of code a fixed number of times. During each iteration a special variable which is defined by the construct, is set to a different value.

The **for** loop

The **for** statement sets the variable to each of the values in the list and executes the block of commands between **do** and **done** statements for each value assigned. Execution ends when there are no more values in the list to assign. A list is one or more words (space delimited).

The visual shows two examples where a **for** loop is used. In the **rm tmp** script a wildcard is used. Before execution of the **for** loop the wildcard will be expanded by the shell. All files (except hidden files) in the **/tmp** directory will be removed.

Instructor Notes:

Purpose — Demonstrate how a `for` loop is structured.

Details — The `for identifier in list` statement is designed to look for some identifier in a list and do a command or series of commands. When the list is exhausted, it falls through to the `done`.

Additional Information — A student may ask what happens if there is no “in list” on the `for` statement. In that situation, the variable will be assigned from each parameter in the parameter list which was pass to this script; when all parameters have been processed, the `for` loop will terminate.

Transition Statement — Let's now see how another loop syntax, the `while` loop works.

while Loop Syntax

```

while      expression
do
    command(s)
done

```

```

$ cat information
x=1
while [[ $x -lt 9 ]]
do
    echo "It is now $(date)"
    echo "There are $(ps -e | wc -l) processes
running"
    echo "There are $(who | wc -l) users logged in"
    x=$(expr $x + 1)
    sleep 600
done

```

© Copyright IBM Corporation 2008

Figure 15-12. **while** Loop Syntax

AU1310.0

Notes:

Introduction

In this construct, an expression is tested for, and as long as this remains true, the body of commands are executed (the commands between the **do** and **done** statements).

The **while** loop

The **while** loop will be executed only if the expression evaluates true. By using the **true** command as the expression in the while statement, it forces the set of commands to be executed until the script is interrupted, for instance with <Ctrl+c>. The **true** command always returns a true result. The **false** command always returns a false result.

The **sleep** command suspends execution of a process for the specified number of seconds.

Instructor Notes:

Purpose — To illustrate how loops can be implemented using the `while` command within scripts.

Details — When the script *information* is executed, it will first display the current date, and then will report on the number of users logged in to the system at the current time and also the number of processes running.

The variable `x` allows the script to execute up to eight times. For example, you may execute the script when you come into the office in the morning, and it will execute once for each of your 8 hours at work.

The `sleep` command is used to suspend the execution of the script for the specified number of seconds. After the `sleep` statement completes, control is passed back to the `while` command where the expression is again tested to ensure that its value still evaluates true. If it is true, then the set of commands are executed again.

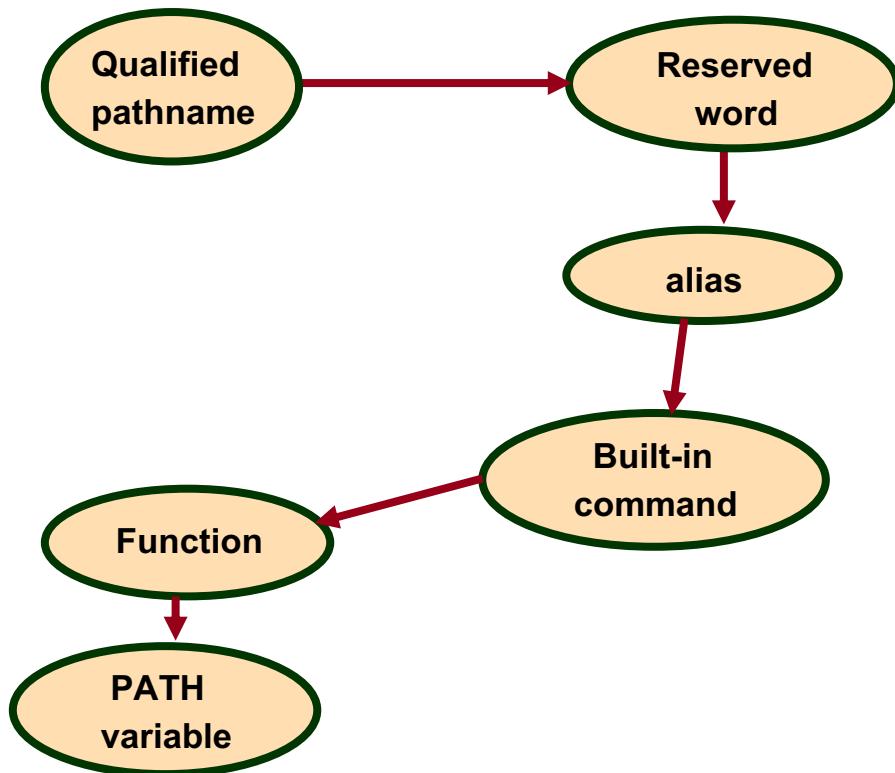
Note that if using the `true` expression with the `while` command, you force the statement to always evaluate to `true`, so the set of commands will execute until the script is interrupted or some other conditional expressions within the loop executes the `exit` statement.

The `test` command is frequently used with the `while` command in the expression part of the construct.

Additional Information —

Transition Statement — Now, let's look at the command search order.

Command Search Order



© Copyright IBM Corporation 2008

Figure 15-13. Command Search Order

AU1310.0

Notes:

Shell command search

This visual describes where the shell looks for the command to be executed when it is ready.

Reserved words are those words that have special meaning to the shell, such as `if`, `then`, `else`, `while`, and so forth.

Built-in commands are those commands that are part of the shell. Examples includes `cd`, `pwd`, `umask`, `read`, and `echo`. If you cannot find a command in the reference manual, try looking under the `ksh` section, or in the AIX 6.1 System User's Guide.

The `PATH` variable is the last thing searched and notice that by default, the current directory is the last directory searched in the `PATH` variable.

Instructor Notes:

Purpose — Describe how the shell searches for a command.

Details — Make sure the students understand how the shell attempts to process each line that is issued at the command line or within a shell script. Review how it breaks apart the line and determines that the first word on the line must be the command name and all remaining words must be parameters to that command name. Once that line has been interpreted, then the shell is ready to invoke the actual command.

Let's take a look at how the system determines where it is actually going to find the command that we want to invoke, and why do we need to know. Quite often, you expect the shell to find a command in a particular place, and because of the search order that it provides, the shell may actually find your command in a different place. When entering a command using the full path name such as **/user/bin/date**, the shell does not have the option of determining where to find the **date** command. You have told it explicitly in which directory to find the **date** command (**/usr/bin**). This is the one way that you can get around the shell search mechanism shown on this page.

If you do not issue a fully qualified path name, but instead issue just the command **date**, then the shell goes through a series of events to determine where to find that command. Do not assume that it looks for the command in your current directory. It looks in many other places before it will look in the current directory. Follow the arrows on the visual to determine the resolution process to find the location of the command.

First, it determines if **date** is a reserved word. If not, is it an alias? If not, is it a built-in command?, and so forth. The student notes define a reserved and built-in command. A function in the Korn shell is a script-within-a-script that you use to define some shell code by name and have it stored in the shell's memory so it can be run at a later time.

Finally, the **PATH** variable is checked. When the shell searches the **PATH** variable, it searches from left to right. This indicates that if you have a **date** command in your current directory and there is a **date** command in **/usr/bin**, it will find the **/usr/bin/date** command first.

The bottom line is to be very careful that the program you are trying to access is not also an alias, or function, or reserved word, and so forth, or the system will find those entities instead of the program that you want the system to find.

Additional Information — Although we have not and will not talk about the **root** user in this class, some students that have more experience with AIX may have noticed a common error when working as **root**; they cannot find commands in their current directory, for example **./date**. If **date** happens to be in **root**'s current directory, to actually find that particular command it will have to be executed as **./date** to tell the shell to look in **root**'s current directory. Most students think that if they are logged in as **root** and if they run the **ls** command and can see the **ls** program in their current directory, that they should be able to execute it. However, if you are working as **root** it will not. This is because at the end of **root**'s **PATH** variable there is no “.” to indicate searching the current directory.

Transition Statement — Let's put it all together and look at a sample customized **.profile**.

Sample .profile

```

PATH=/bin:/usr/bin:/etc:$HOME/bin:.
PS1='$_PWD => '
ENV=$HOME/.kshrc

if [ -s "$MAIL" ]
then
        mail
fi

echo "Enter Terminal Type (Default:ibm3151):\c"
read a
if [ -n "$a" ]
then
        TERM=$a
else
        TERM=ibm3151
fi

echo "It is now $(date) "
echo "There are $(ps -e | wc -l) processes running"
echo "There are $(who | wc -l) users logged in"

export PATH ENV TERM PS1

```

© Copyright IBM Corporation 2008

Figure 15-14. Sample .profile

AU1310.0

Notes:

Customizing the environment

The **PATH** variable sets up the directory search path for commands and executable shell scripts. This example only includes **/bin**, **/usr/bin**, **/etc**, **\$HOME/bin** and the current directory (**.**).

The **PS1** variable sets up the primary prompt string for the command line shell prompt. In this example it will be the current directory followed by an arrow. For example, **/home/team01 => .**

The **ENV** variable sets up the directory and file for Korn shell customization such as alias.

The **if-then** construct with **MAIL** checks for the existence of **mail** and if there is some, the **mail** command will automatically execute and put the user immediately into a mail session.

The next section of the **.profile** example provides for an interactive setting of the terminal type by prompting the user to provide a terminal type. If the user provides a terminal type, then the `TERM` variable will be set to that value. Otherwise, if no input is provided, the `TERM` variable will be set to a default of `ibm3151`.

Then the user is shown the current date, number of processes currently running, and the number of users currently logged in.

The last part of this script exports the variables that have been set in order to make them available to child processes.

Instructor Notes:

Purpose — The `.profile` is the file that the user will use to customize their environment. So in this last example all the ideas from previous sessions as well as this session are used to illustrate a customized `.profile`.

Details — At this stage there are no new ideas, however, go quickly through each customized parameter. Mention the following:

PATH	will display the current path
PS1	The prompt
ENV	remind the user why this variable has to be used

The next few lines have been written for new incoming mail. The `if` statement is used to test the file where a user's mail messages are sent. If this file is not empty (that is, new mail items have arrived) then the `mail` command will automatically be invoked. So every time the user logs in, if there is mail they will be forced to read it before continuing.

The next `if` statement is used to set the `TERM` variable. The user is prompted for a value. (Note here that the `\c` is used with the `echo` statement. This ensures that the cursor remains on the same line as the text that is displayed for the user.) When the user inputs a value, this will be stored in the `a` variable and will be used as the `TERM` type. However, if the user does not input a value, then the default value of `ibm3151` will be used. Point out that this is not a complete script because what happens if a user enters text which is incorrect or not valid for that terminal type. This could have major implications. This part of the script can obviously be modified further.

The last part of the script prints out information about the system; that is, the current date, the number of processes running and the number of users logged in.

Additional Information —

Transition Statement — Let's cover a few checkpoint questions before going to the lab exercises.

Checkpoint

1. What will the following piece of code do?

```
TERMTYPE=$TERM  
if [ $TERMTYPE != "" ]  
then  
if [ -f /home/team01/customized_script ]  
then  
    /home/team01/customized_script  
else  
    echo No customized script available !  
fi  
else  
echo You do not have a TERM variable set !  
fi
```

2. Write a script which will accept two arguments, multiply them together, and display the result.

© Copyright IBM Corporation 2008

Figure 15-15. Checkpoint

AU1310.0

Notes:

Instructor Notes:**Purpose —****Details —**

Checkpoint Solutions

- What will the following piece of code do?

```
TERMTYPE=$TERM
if [ $TERMTYPE != "" ]
then
if [ -f /home/team01/customized_script ]
then
    /home/team01/customized_script
else
    echo No customized script available !
fi
else
echo You do not have a TERM variable set !
fi
```

The script will set a variable **TERMTYPE** to the value of the **TERM** variable. In the if statement the **TERMTYPE** variable will be tested to see if it is not empty. If it is not, then a second check will be carried out to ensure that the **/home/team01/customized_script** file is an ordinary file. If it is then it will be executed.

(For our example we will assume that this file contains some extra customized features.) If this file is not an ordinary file, then a message will be sent to the user stating this. If the initial test fails, that is, the **TERMTYPE** variable is empty, then again a message will be sent to the user.

- Write a script which will accept two arguments, multiply them together, and display the result. **expr \$1 * \$2**

© Copyright IBM Corporation 2008

Additional Information —**Transition Statement —** Let's do a lab.

Exercise: Additional Shell Features



© Copyright IBM Corporation 2008

Figure 15-16. Exercise: Additional Shell Features

AU1310.0

Notes:

After completing the exercise, you will be able to:

- List common constructs used in writing shell scripts
- Create and execute simple shell scripts

Instructor Notes:

Purpose — To introduce the next exercise.

Details — Describe what students will learn in the lab.

Additional Information —

Transition Statement — Before moving on to the next unit, let me review the important points from this unit.

Unit Summary

- **Positional parameters** are used to pass to scripts the command line arguments.
- To **test** for a **particular condition** the test command can be used.
- The **test** feature can be coupled with the **if** statement to control the **flow of a program** and allow for conditional execution within scripts.
- The **read** command can be used to implement **interactive scripts**.
- The **while** command is used to **Maintain loops** until a condition fails.
- The **for** command allows to **repeat a section of code** a fixed number of times.

© Copyright IBM Corporation 2008

Figure 15-17. Unit Summary

AU1310.0

Notes:

Instructor Notes:

Purpose — To summarize this unit's key points.

Details —

Additional Information —

Transition Statement — Let's move on to a new unit.

Unit 16. The AIX Graphical User Interface

Estimated Time

01:30

What This Unit Is About

This unit provides an overview of the AIXwindows and CDE environment.

What You Should Be Able to Do

After completing this unit, you should be able to:

- List the advantages of working in an AIXwindows environment
- Explain the AIXwindows client/server model
- Start AIX windows and initiate X Clients
- Start an `aixterm` window
- Display remote clients on your system
- Describe the goal of the CDE environment
- Use the various CDE components

How You Will Check Your Progress

- Student activity
- Checkpoint questions
- Exercise

References

SC23-2793

Common Desktop Environment 1.0 User's Guide

Unit Objectives

After completing this unit, you should be able to:

- List the advantages of the AIXwindows environment
- Explain the AIXwindows client/server model
- Start AIXwindows and initiate X Clients
- Display remote clients on your system
- Introduce the Common Desktop Environment (CDE)
- Describe commonly used CDE components

© Copyright IBM Corporation 2008

Figure 16-1. Unit Objectives

AU1310.0

Notes:

Instructor Notes:

Purpose — To introduce the unit objectives to the students.

Details —

Additional Information —

Transition Statement — First, a bit of history about the X Window system.

The X Window System

- Network-based graphics subsystem
- Developed at MIT in 1984 (Project Athena)
- Designed initially for UNIX-based systems
- Important design aspects
 - Used client/server model
 - Network aware (works over TCP/IP)
 - Protocol is independent of operating system and graphics device
- AIX 6.1 includes X11R7.1
 - Backwards compatible with all prior releases
- IBM's port of the X Window System is called "AIXwindows"

© Copyright IBM Corporation 2008

Figure 16-2. The X Window System

AU1310.0

Notes:

History of X

The *X Window system*, called **X** for short, is a network-based graphics system that was developed at MIT in 1984. It was designed as a generic, UNIX-oriented basis for graphical user interfaces (GUIs). Prior to X, the only way to communicate with a UNIX system was using commands in an ASCII environment.

In 1987, a group of vendors and researchers formed the X Consortium to continue work on this windowing system. X version 11 (X11) was released in 1987 and continues to be the version of X that is used. There have been several releases of X, the most current being release 7 (2005). The X Consortium code is freely available and will run on most UNIX architectures.

AIXwindows

AIXwindows is AIX's windowing system. AIXwindows includes the X Window system, OSF Motif, and the Common Desktop Environment. The Motif window manager (mwm) is used to control such things as the size and position of the windows.

Instructor Notes:

Purpose — Introduce AIXwindows.

Details — AIX V3.2.5, V4.1, and V4.2 use X11R5. AIX V4.3, AIX 5L V5.1, AIX 5L V5.2, and AIX 5L V5.3 incorporate X11R6. AIX 6.1 introduces X11R7.1. AIXwindows supports both 2D and 3D products. Our focus in this class will be on the AIXwindows 2D support.

AIX continues to include compatibility libraries for X11R3, X11R4, X11R5, and X11R6.

Additional Information — For information on changes and enhancements for each of the X Window system releases, see the X Consortium Web site, located at www.x.org.

CDE is built upon Motif 1.2.

AIXwindows also includes 3D support. The application programming interfaces (APIs) that support 3D are graphics and Graphics Library. AIX V4.1 and V4.2 also support PEX.

Transition Statement — Let's take a moment to look at what AIXwindows can do for us in the AIX environment.

What is AIXwindows?

- AIXwindows is IBM's enhancement to the X Window system and Motif
- The windows enable you to work with multiple items simultaneously
- Provides window functions such as: opening, sizing, and moving
- Provides the capability to manage local and remote displays

© Copyright IBM Corporation 2008

Figure 16-3. What is AIXwindows?

AU1310.0

Notes:

AIXwindows: an implementation of X Window

AIXwindows provides a graphical user interface environment. It also provides a graphical desktop that hides the low-level complexities of the operating system.

AIXWindows uses a client/server environment. The advantage here is that the graphics application can run on one system, yet display its output on another system.

Instructor Notes:

Purpose — Explain that AIXwindows is IBM AIX's implementation of the X Window System.

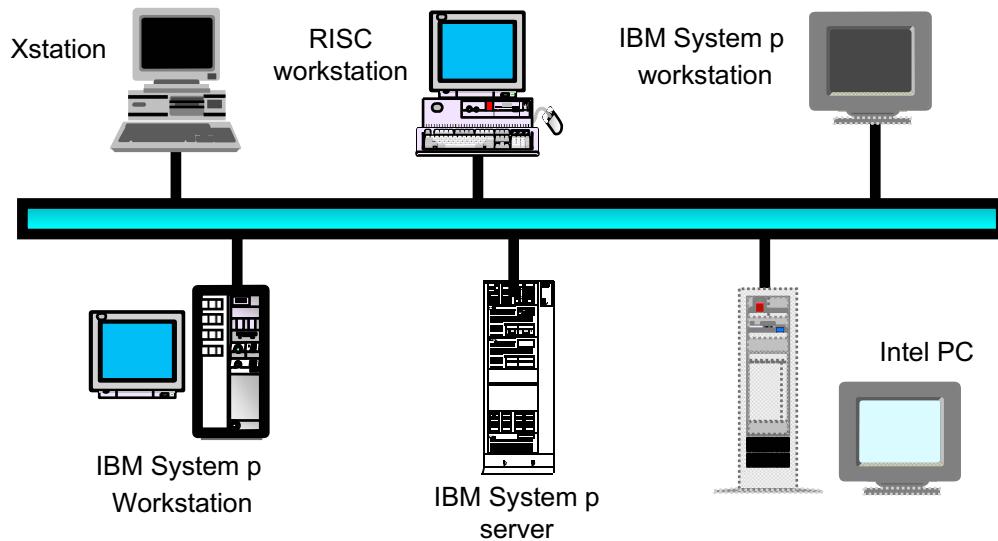
Details — Be sure to mention that the X Window system (and thus AIXwindows) runs in a client/server environment.

Additional Information —

Transition Statement — Let's take a look at how AIXwindows can work in a networked environment.

An X Window System Network Configuration

- Networked workstations and file servers
- Heterogeneous environment
- A client/server environment



© Copyright IBM Corporation 2008

Figure 16-4. An X Window Network Configuration

AU1310.0

Notes:

AIXwindows in a network environment

Above is an example of a network of machines running X. The X Window system is platform independent. It allows a display and keyboard attached to one system to use programs running on a completely different type of system.

The X Window system function is split into two parts: the terminal support and the application support.

Typically, the application support runs on a UNIX system. The terminal support can run on the same UNIX system, on a remote UNIX system, on an X Station, or even on a non-UNIX PC. This is why the X Window system is commonly referred to as a *networking* window system.

This is another way of saying that it supports the *client/server* environment. The system providing the application support is known as the *client* and the system that supplies the

terminal support is known as the *server*. In many cases, both the client and the server will be the same system.

Instructor Notes:

Purpose — To show the environment that can run AIXwindows and other X Window system products.

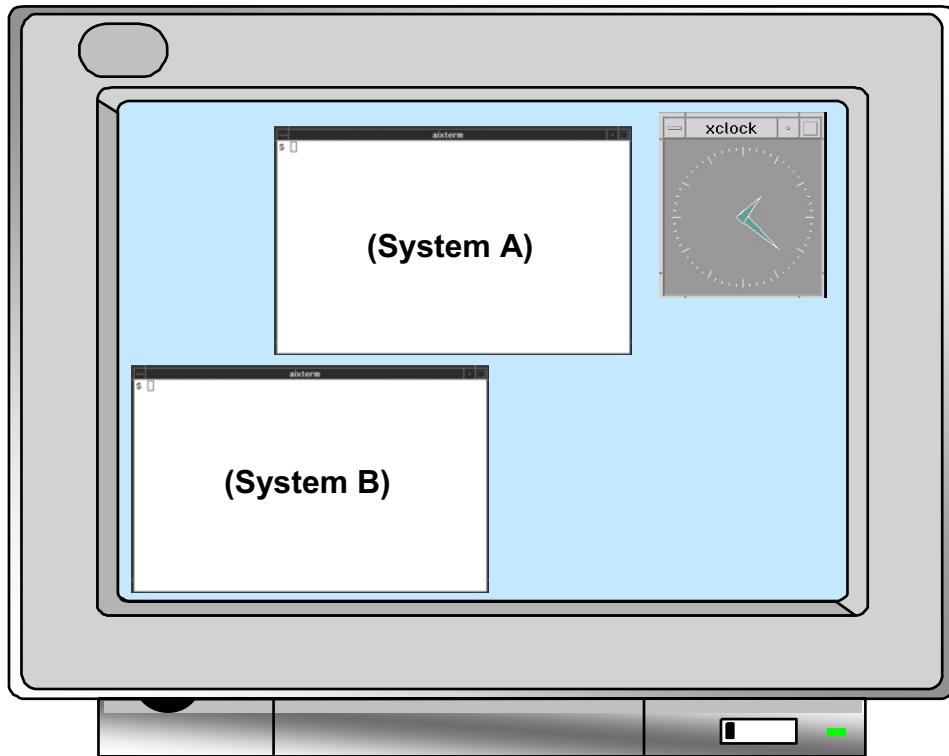
Details — This visual shows a common customer environment with many types of computer systems networked together. The visual shows three IBM System p systems in the network, but these could really be any type of RISC processor. In this environment, an AIXwindows application could run on one of the IBM System p processors. However, a user can sit at any one of the other systems in the network and run an X Window application.

An Xstation is another way to provide LFT support on an IBM System p. Most System p systems support only one graphics adapter, and thus only one high-resolution monitor (LFT). An Xstation is a way to provide additional high-resolution displays without having to purchase another System p system. In the marketplace, these displays are known as X Terminals. IBM's were known as Xstations. In most cases, they will provide better resolution than a PC.

PCs can use Hummingbird's eXceed for Windows, to provide the X terminal support. Another one is provided by the Cygwin Project (www.cygwin.org). There are also many other emulators on the market that can be used.

Transition Statement — Let's take a closer look at the X Window client/server model.

The Client/Server Environment



© Copyright IBM Corporation 2008

Figure 16-5. The Client/Server Environment

AU1310.0

Notes:

X Clients

In the world of X, the *client* is the application that is running and needs to display graphics to a user. In the above graphic, the two AIX window screens and the clock are considered clients. One of the terminal screens (System A) could be a client from the local system while the second terminal screen (System B) could be a client from another system in the network.

The X Server

The *server* runs on a computer with bitmapped (graphics capable) terminals. Clients send display information to the X Server. Clients receive keyboard and mouse input from the associated X Server. X Servers are event driven, that is, they respond to requests from clients and to actions from the user. X Servers used by an X Client do not have to be on the same platform as the X Client.

Instructor Notes:

Purpose — Explain the *client/server* relationship in the X Window world.

Details — The client/server relationship in the X Window world may seem backwards to most users. With X, the clients are device-independent programs that wish to have graphics displayed. The servers are device-dependent programs that control the screen and input. In this environment, the user interacts with the X Server and not the X Client.

Another way to look at it is that the X Client is the application that is running and needs a way to get the information to and from the user. The X Server has a display it can write on and a keyboard and mouse that it can accept input from.

In the graphic, the two terminal screens and the clock are considered *X Clients*. Be sure to mention that one of the terminal screens (System A) could be a client from the local system while the second terminal screen (System B) could be a client from another system in the network. We will discuss the concept of running remote clients later in this unit.

Transition Statement — Let's further discuss the characteristics of an X Client.

X Clients

- **X Clients** are the **applications** which the user runs under the X window system:

Examples: **aixterm**, **xterm**, **xclock**, **xcalc**, **xwd**, **mwm**

- **X Clients** can be started from the **command line** or from special **startup files**

- Most X clients share the **same options**:

-bg color	Color for the window background
-bd color	Color for the window border
-bw number	Width in pixel of the window border
-display hostname:number	Identifies host server name and X Server display number
-fg color	Color for the window foreground
-fn font	Normal sized text fontset

© Copyright IBM Corporation 2008

Figure 16-6. X Clients

AU1310.0

Notes:

Common X Clients:

xterm	The standard terminal emulator included in the X Window system
aixterm	The IBM AIX terminal emulator
xclock	Displays a clock
xcalc	Scientific calculator
xwd	Dumps the image of an X client
mwm	The Motif window manager

Standard X Client command line options include:

-bg Color	Specifies the color for the window background.
-bd Color	Specifies the color for the window border.

-bw Number	Specifies the width in pixels of the window border.
-display Name:Number	Identifies the host server name and the X Server display number where the command is to run. If this is not specified, the client program gets the host name and display number from the DISPLAY environment variable.
-fg Color	Specifies the color for the window foreground.
-fn Color	Specifies the normal sized text fontset.

Instructor Notes:

Purpose — Further describe the X Client.

Details — The X Client `aixterm` is unique to AIXwindows. Using an `xterm` or `aixterm` window, you can do anything you might do on a regular terminal: enter commands, compile programs, edit programs, and so forth.

Additional common command line options:

- geometry <Geometry>** Specifies the location and dimensions of a window. The default is 80x25+0+0.
- iconic** Displays the window as an icon rather than the normal window when the window is created.
- xrm <String>** Specifies a resource string to be used.

Additional Information — The `xwd` command can be used to dump the image of an X client window to a file. This file can then be read by other utilities that perform functions like re-displaying and printing the image. This command uses the `-out` option to indicate where the screen image will be dumped.

Transition Statement — Now, let's talk further about the X Server.

The X Server

- Each X Server:
 - Controls one keyboard, one mouse, and one or more screens
 - Allows simultaneous access by several clients
 - Performs basic graphic operations
 - Provides information such as fonts and colors
 - Routes keyboard and mouse input to the correct clients

© Copyright IBM Corporation 2008

Figure 16-7. The X Server

AU1310.0

Notes:

Introduction

The X Server is a critical part of the X Window System.

The X Server

The primary role of the X Server is to intercept keyboard and mouse input (called *events*), and direct that information to the appropriate X Client. It also receives information back from the client and then redraws the graphics screen. The X Server also handles the resolution and color depth of the screen.

The X Server runs on the machine where the graphics monitor, keyboard, and mouse are attached to. This can be an AIX machine, a PC with X Server software, or a dedicated device (like an IBM Xstation).

Instructor Notes:

Purpose — Further define the purpose of the X Server.

Details — An Xstation is a dedicated device. It has a fully functioning microprocessor, but the only application it runs is X Server code and communications code (such as `tftp` and TCP/IP).

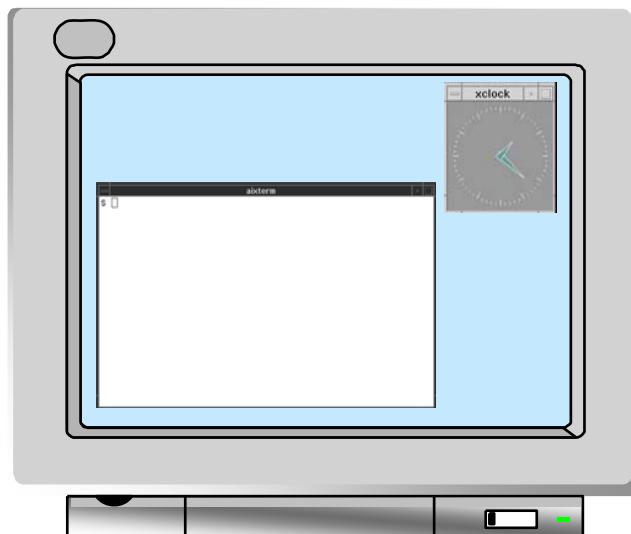
Additional Information — Sometime we want to use a PC as an X Server, similar to how we use an Xstation. To do this one would need to install and configure some sort of X Server software on the PC. One popular type of software which does this is Hummingbird eXceed.

Transition Statement — Let's now take a look at using the AIXwindows system.

Starting X

\$ startx

- Starts the X server, Motif Window manager (`mwm`), and the `aixterm` and `xclock` clients



© Copyright IBM Corporation 2008

Figure 16-8. Starting AIXwindows

AU1310.0

Notes:

startx command

Use the `startx` command to start the AIXwindows environment. This command determines the type of X Server being used and then starts that X Server. If using a workstation (that is, not an Xstation), `startx` will execute the `xinit` command.

What is started

By default, `startx` will also start three X Clients: an `aixterm`, the `xclock`, and the Motif window manager (`mwm`). The `aixterm` can be used like any other terminal to enter commands, edit files, and compile programs.

Default configuration

The look and feel of the initial screen started by `startx` can be tailored. For example, you may wish to display two aixterm screens as well as a scientific calculator. This will be discussed in further detail in the next unit.

Startup errors

Any errors encountered during the AIXwindows startup process will be sent to a file called **\$HOME/.xerrors**. If the file does not exist, it will be created automatically.

It is also possible to set an environment variable `XERRORS`. This variable should be set if you wish to route errors to a file other than **\$HOME/.xerrors**.

Instructor Notes:

Purpose — Discuss the initial screen displayed by the `startx` command.

Details — The graphic shows the initial screen displayed by the `startx` command. Actually, `startx` is a shell script. Among other things, it determines the type of terminal screen you are working on. If you have an LFT attached directly to an IBM System p system, the `startx` shell script will execute the `xinit` command. In this environment, running the `xinit` command will produce the same result. On an Xstation, you must use the `startx` command as `xinit` will not work.

Point out that when the students actually run the `startx` command themselves, they will first see just the windows (no borders) displayed. This is the X Window system starting up. Borders will be placed around the windows once the Motif window manager has run.

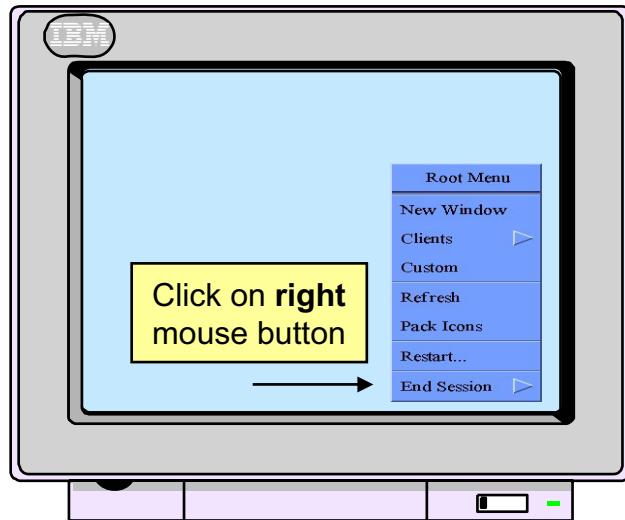
Additional Information — AIXwindows uses the Motif window manager (mwm) as its window manager. If running CDE, the window manager that is used is the Desktop windows manager (dtwm).

The proper way to exit your session is to use the root menu that we will be covering later. In some cases, such as the Common Desktop Environment, the `<Ctrl+Alt+Backspace>` key sequence is disabled for security reasons.

Transition Statement —

Stopping X

- Click the desktop background (known as the **root window**) with the **right mouse button**.
- Select **End Session**.
- If the X server will not exit, use **<Ctrl><Alt><Bksp>**



© Copyright IBM Corporation 2008

Figure 16-9. Stopping X

AU1310.0

Notes:

Exiting AIXwindows

Before exiting AIXwindows, all running applications need to be terminated.

The correct way to exit AIXwindows is to right-click the root window (the display background). From the menu, select **End Session**.

If the AIXwindows session is unresponsive, use the key sequence **<Ctrl><Alt><Backspace>** on the LFT to terminate the X Server and return to the command prompt. Beware that any applications running in the X session will also terminate.

Instructor Notes:

Purpose — To explain how to stop the AIXwindows session

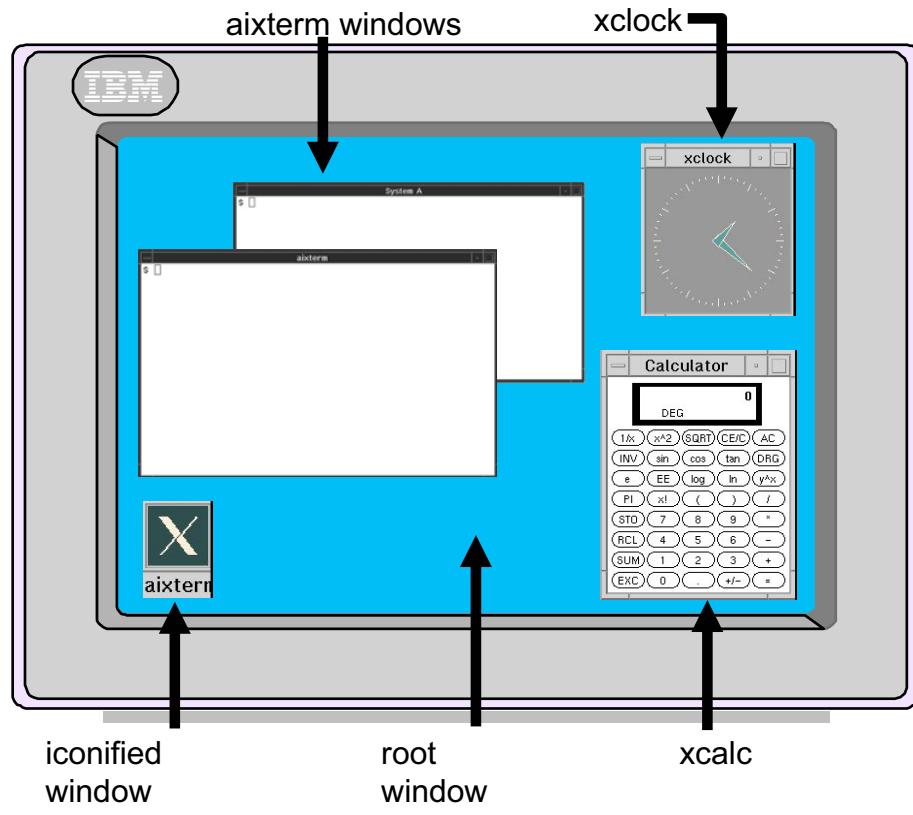
Details — Stopping the X Server with the **<Ctrl><Alt><Backspace>** key sequence will cause all X Clients to terminate, and may result in application data loss. As such, this should only be done as a last resort, if the X Server cannot be terminated from the window manager.

Always terminate the applications using the normal menu options.

Additional Information —

Transition Statement — Let's explain the AIXwindows display.

An AIXwindows Display



© Copyright IBM Corporation 2008

Figure 16-10. An AIXwindows Display

AU1310.0

Notes:

Anatomy of the AIXwindows display

The above graphic illustrates what an AIXwindows display can look like. Certain windows will accept information from the user, such as the two **aixterm** displays. Some windows simply display information, such as the **xclock** and the **xcalc**.

Windows can be *iconified* in order to clear the clutter off the screen. The shaded area that fills the entire screen is called the *root window*. The *root window* actually has its own menu which can be used to start additional windows as well as tailor the AIXwindows environment. This *root menu* will be discussed in more detail later.

Instructor Notes:

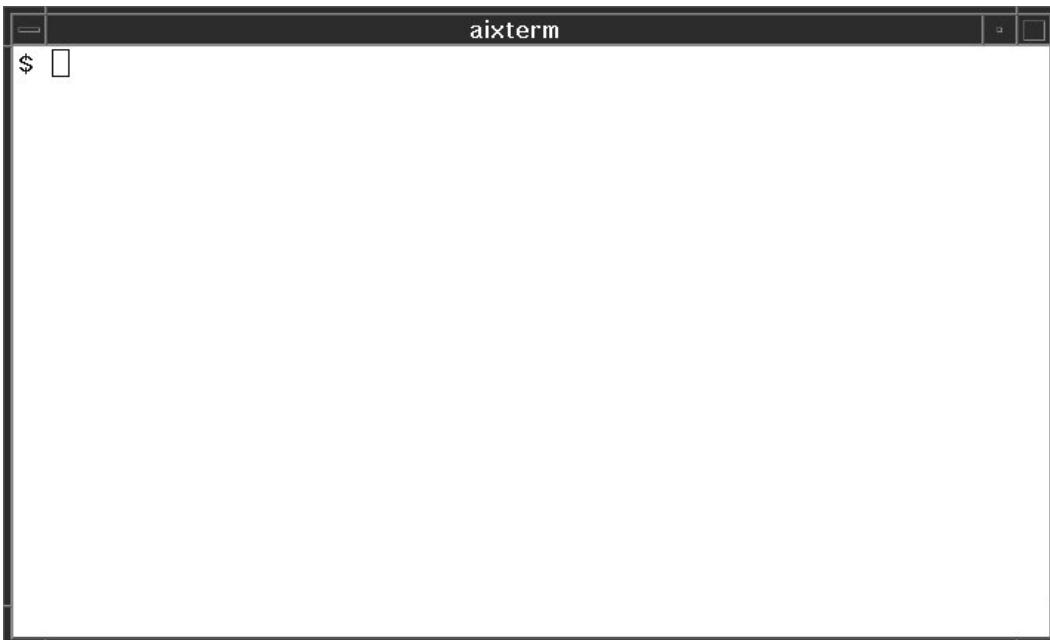
Purpose — Discuss the anatomy of the AIXwindows display.

Details — We will discuss how this display can be tailored later in the class.

Additional Information —

Transition Statement — Let's look at one of the most common X clients, the `aixterm` terminal emulator.

The aixterm Window



An **aixterm** can be started in two ways:

- ➔ In a shell, by using the command **aixterm** &
- ➔ Display the **Root Menu** and select **New Window**

© Copyright IBM Corporation 2008

Figure 16-11. The aixterm Window

AU1310.0

Notes:

Terminal window

Use the **aixterm** window to enter AIX commands just as you would from a character-based ASCII screen.

It is possible to create additional **aixterm** windows when using AIXwindows. This can be done in two ways:

- From an open **aixterm**, enter the **aixterm** & command.
- Move the mouse pointer to the root window and press the right mouse button. When the root menu appears, choose **New Window**.

Moving data between windows

The **aixterm** window allows text to be copied and pasted to another part of the window or even to another window. To accomplish this, position the mouse pointer at the first letter you want to copy and drag the left mouse button over the text to be copied. The

text will be highlighted. When you release the button, the highlighted text is copied into a hidden buffer and the highlighting disappears. Move the pointer to where you want to place the copied text and press the center mouse button. The text is then copied from the buffer into the new window (even if the window is not active).

aixterm scrollbars

It is sometimes helpful to create a scrollbar for the **aixterm**. To do this, place the pointer inside the window, hold down the <Ctrl> key and press the center mouse button. This will display the *Modes Menu*. Click the left mouse button on the *Scrollbar* entry and a scrollbar will appear on the right side of the window. Once in the scrollbar area, use the left mouse button to move the text up and the right mouse button to move the text down.

Exiting aixterm

The **aixterm** window can be closed several ways:

- Type **exit** or <Ctrl-d>
- Double-click at the upper left of the window frame
- Single-click at the upper left of the window frame and then click close

Instructor Notes:

Purpose — Briefly explain the `aixterm`.

Details — Most users of AIX will spend most of their time using the `aixterm` windows. There are many functions available with these windows, but the most commonly used are copy and paste, adding a scrollbar and closing the window. The student notes explain how to accomplish these functions. Students will also perform these tasks in the machine exercise.

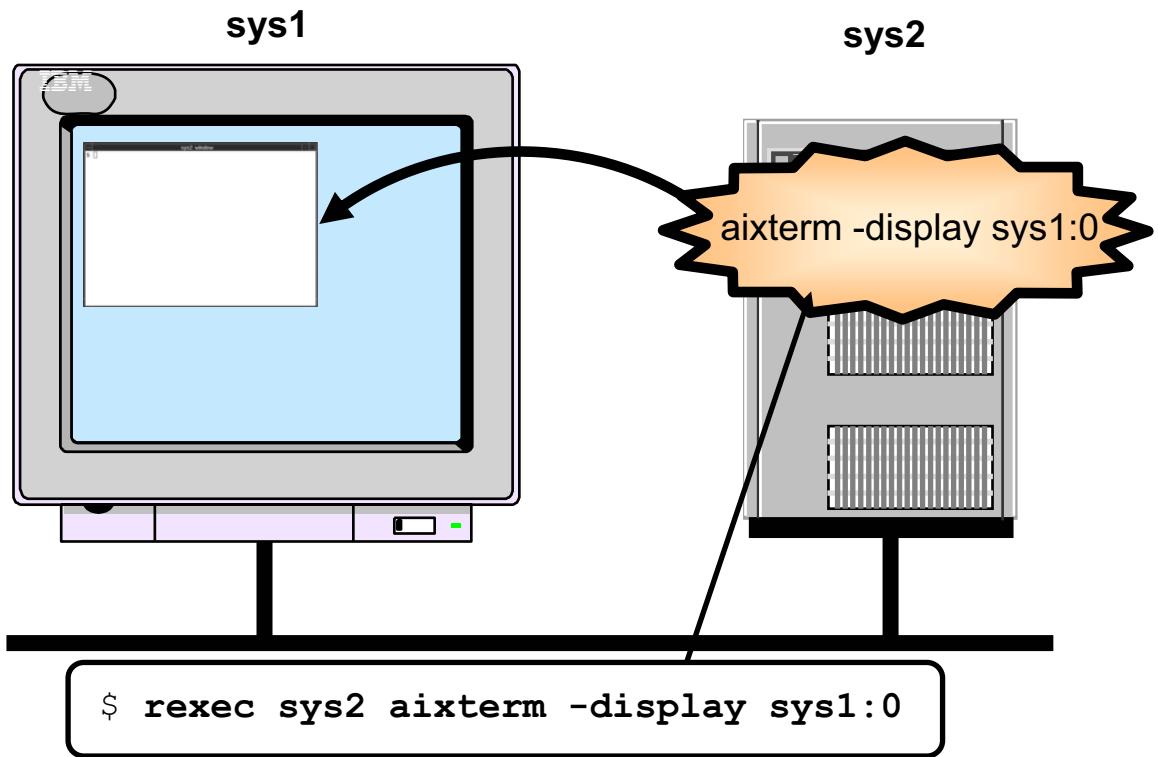
By default, the scrollbar is placed to the right of window. If using the `aixterm` command to create an `aixterm` window, it is possible to specify both the `-sb` and `leftscroll` options to place the scrollbar on the left.

When using a two button mouse, pressing both button simultaneously provides the same functions as the center button on a three button mouse.

Additional Information —

Transition Statement — Let's take a look at the Common Desktop Environment.

Running a Client on Another System



© Copyright IBM Corporation 2008

Figure 16-12. Running a Client on Another System

AU1310.0

Notes:

Introduction

As mentioned earlier, AIXwindows uses the client/server model. As a review, the client is the graphics application (such as `aixterm` or `xcalc`) while the server is the display supporting the application. In many cases, the client and server run on the same system.

However, with AIXwindows it is possible to run a client on a remote system in the network, yet display the application window on your own screen. You continue to enter commands in the window and use the mouse even though the actual process is running on another system. This arrangement gives you access to software programs that are running on remote systems.

DISPLAY variable

In order to run a client remotely and display its results locally, it is necessary to tell the client process where to display its window. AIXwindows applications use the value of the `DISPLAY` environment variable to indicate the name of the server (that is, where the client should display its output). This value can be overridden using the `-display` option when starting a client. The display value is usually set to something like `:0.0` for local servers or `sys1:0.0` to have the client display its output on a remote server.

Example

In the above visual, `sys2` runs the client application, while `sys1` needs to display the output of the application. For our example, the application is `aixterm`, but it could be any AIXwindows application. The graphic shows two ways to display `sys2`'s output on `sys1`. In both cases, the `-display` option indicates the name of the server. `rexec` is a TCP/IP command that sends a command to a remote system for execution.

Instructor Notes:

Purpose — Some users may find it necessary to run a client on another system, yet display the output on their own system. We will discuss this here.

Details — In practice, if the `DISPLAY` variable is set, it is usually set to the local system's TCP/IP *hostname*. If this value needs to be overridden for a specific X Client, then use the `-display` option for that client application.

The display option is usually set to something like `:0.0` for local servers and `sys1:0.0` for remote servers. The part before the colon specifies the TCP/IP *hostname* or IP address of the server. The part after the colon specifies the server number and display number (the display number is optional). Unless you run multiple servers on a machine or have multiple displays controlled by one server, these values will be set to zero. Naturally, TCP/IP would need to be configured in this environment.

In our example, it is necessary to run the TCP/IP `rexec` command from `sys1` to indicate the name of the system where the `aixterm` command is to be run. For those unfamiliar with `rexec`, think of it as a way to perform a `telnet` and then an execution of a command at the remote system. `rexec` will prompt for an ID and a password in order to access the remote system.

Additional Information —

Transition Statement — The server can limit which remote hosts have the ability to display their output on the server's display. That is the purpose of the `xhost` command. We will look at the `xhost` command next.

The xhost Command

The **xhost** command **adds** and **deletes** hosts on the list of machines from which the X Server **accepts connections**:

```
xhost [ + | - ] [ hostname ]
```

\$ **xhost + moon**

Allow moon to start
X clients

\$ **xhost - pluto**

Deny pluto to start X
clients

\$ **xhost +**

Allow all hosts to
start X clients

© Copyright IBM Corporation 2008

Figure 16-13. The xhost Command

AU1310.0

Notes:

Using the xhost command

Initially, the X Server only allows connections from X Clients running on the same machine, or clients running on systems listed in the file **/etc/X0.hosts**.

The **xhost** command must be executed on the machine to which the display is connected. A host can be removed from the access list by using the command:

```
xhost - hostname
```

Similarly, a host can be added by using the command:

```
xhost + hostname
```

Specifying the command **xhost +** allows all hosts to connect to the X Server effectively disabling the host access control system.

The **xhost -** allows no other hosts to connect to the X Server.

Entering the command **xhost** shows the names of the hosts allowed access to the X Server.

The **-display** option (discussed on the previous page) designates which X Server an application wants to talk to, and the **xhost** command determines if that X Server is willing to talk to that X Client application.

Instructor Notes:

Purpose — Explain how a server can limit which clients they are willing to talk to.

Details — If a list of hosts is placed in file **/etc/X?.host** (where ? is the number of the server; usually 0) then those hosts will be authorized by default.

Additional Information —

Transition Statement — Let's look at another method of limiting access to the X server, this time through the **xauth** mechanism.

The **xauth** Command

The **xauth** command is used to display and edit the authorization information used in connecting to the X server.

Uses a cryptographic key (called a **cookie**)

Allows for per-user access control to the X server

Overrides the **xhost** mechanism.

```
hostA$ xauth list
hostA:0 MIT-MAGIC-COOKIE-1 566e6b7a786543387746464337396c67
hostA$ xauth extract xauthfile.txt hostA
hostA$ scp xauthfile.txt team02@hostB:xauthfile.txt
```

```
hostB$ xauth merge xauthfile.txt
hostB$ aixterm -display hostA:0 &
```

© Copyright IBM Corporation 2008

Figure 16-14. The **xauth** Command

AU1310.0

Notes:

The X Server can also limit access to X Clients by using a cryptographic key. This allows access to be granted by **user**, rather than by **host**. If a user has the cryptographic key, they are allowed access to the server.

These keys are stored in the **.Xauthority** file located in the X Server's process owner's home directory. For example, if user **team01** started an X Server, the server's cryptographic key would be located in **/home/team01/.Xauthority**.

To assign the key to a remote user, it needs to be extracted to a file. The file is then transferred to the remote host, and merged into the remote user's **.Xauthority** file. The remote user can then display X Clients to the X Server.

The extraction and merging of the key is done through the **xauth** command. The **xauth** command has a number of common commands:

- **add**: Adds a key to the **.Xauthority** file. The display name, key protocol type, and hex key is required.
- **extract**: Write the specified X Server display key to the specified file.

- **generate**: Generate a new cryptographic key.
- **list**: Display all keys and corresponding displays.
- **remove**: Remove the specified keys from the `.xauthority` file.

Note: The **xauth** command will override any **xhost** client settings.

Instructor Notes:

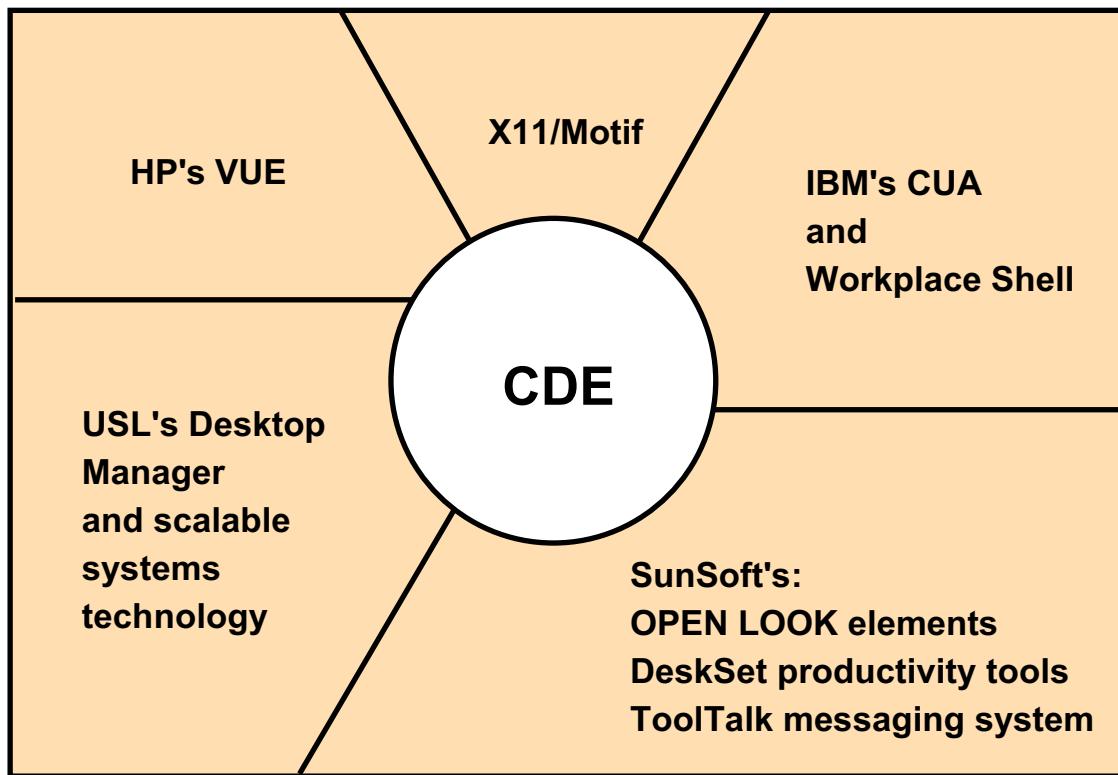
Purpose — To explain how the **xauth** mechanism works

Details —

Additional Information —

Transition Statement — Let's now talk about the Common Desktop Environment, starting off with a little history.

Common Desktop Environment (CDE)



© Copyright IBM Corporation 2008

Figure 16-15. Common Desktop Environment (CDE)

AU1310.0

Notes:

Introduction

The Common Desktop Environment (CDE) provides a common user interface for the UNIX environment.

Where did CDE come from?

Even though the X Window system provided a common base and many UNIX systems used Motif, there were still many incompatible and competing directions being followed. Much of the UNIX open system strength was being lost. There was also the issue of MS-Windows presenting a unified and competitive alternative to UNIX. As a result, the major UNIX vendors realized that they had to agree to a common direction for UNIX and X.

In 1993, a group of vendors formed the Common Open Software Environment (COSE). The focus of this group was to support a common user interface to UNIX. This led to a

set of specifications called the CDE based on windowing and object technologies from Hewlett-Packard, IBM, SunSoft, Novell and OSF.

CDE is a set of specifications based on technologies from:

- IBM's Common User Access standard and shell
- Hewlett-Packard's Visual User Environment desktop
- OSF Motif
- Novell USL's UnixWare clients/desktop manager
- SunSoft's OPENLOOK and DeskSet

CDE has been the desktop interface included in AIX since v4.3.

Instructor Notes:

Purpose — Explain what the CDE is and where it came from

Details — CDE is also supported by HP-UX, Solaris, SCO UNIX and USL UNIX.

AIX V3.2.5 supported a user interface called the AIXwindows Desktop (XDT). This was replaced by CDE in AIX V4.3.

Additional Information —

Transition Statement — Let's begin to look at the components of CDE.

The Components of the CDE Desktop

- The **Login** Manager
- The **Front Panel**
- The **Style** Manager
- The **File** Manager
- The **Application** Manager
- **Personal** Applications
- The **Help** Manager

© Copyright IBM Corporation 2008

Figure 16-16. The Components of the CDE Desktop

AU1310.0

Notes:

CDE components

- Login Manager: Authenticate and initiates the desktop
- The Front Panel: The user interface and launcher
- The Style Manager: Used to customize the desktop
- The File Manager: GUI to work with files
- The Application Manager: For managing applications
- Personal Applications: Can be used to access a `dtterm`
- The Help Manager: Hypertext help information

These components will be covered in more detail in this unit.

Instructor Notes:

Purpose — Introduce the components of CDE.

Details —

Additional Information —

Transition Statement — We will first look at the Login Manager.

The Login Manager



Figure 16-17. The Login Manager

AU1310.0

Notes:

The Login Manager

The login manager prompts for the user name, and then for the password. The password does not appear on the screen.

The process is intuitive but an online Help is available for novice users.

The Options button allows the user to:

- Select which language to use.
- Choose whether to use a regular or fail-safe session.
- Return to command line mode (only on LFT display).
- Restart the login manager.

Your system may automatically display this login window. If not, you can access CDE from the command line by typing `xinit /usr/dt/bin/Xsession`.

Instructor Notes:

Purpose — Introduce the Login Manager.

Details — Show students how to log in to the CDE. In many environments, the graphic shown is the one that users use to log in to the AIX system. If a user has a graphics terminal, but does not log in to CDE automatically, they can do so by typing at the command line: **xinit /usr/dt/bin/Xsession**. Xsession will not start the CDE login prompt should, but rather will start the CDE desktop that the user sees normally after logging in through the Login Manager.

The CDE login screen includes an **Options** button. This button provides several options:

- Select a language to use. The default language for a system is set by the root user. Options enables a user to access different languages. Choosing a language in the Options menu sets the LANG environment variable for the sessions. The default language is restored at the end of the session.
- A fail-safe session starts only a single **aixterm** window. This may be helpful if the user needs to access a single window to execute several commands before logging into the CDE. Exit the fail-safe **aixterm** window by typing **exit**.
- It is also possible to start a command line session. This brings the terminal up in ASCII mode; no windows are started. Certain types of configurations, such as X Stations (X Terminals) do not provide a command line login option.

Be sure to mention to the students that as soon as they log in to CDE, a number of CDE-related directories and files will be added to their **\$HOME** directory. The primary directory that will hold the user's CDE information will be **\$HOME/.dt**. In this directory are a number additional files and directories related to the user's CDE environment. The files and directories are too numerous to cover at this point. However, they should be aware that one of the files that gets placed in their **\$HOME** directory is **.dtprofile**. This is the file we will discuss next.

Additional Information — If accessing CDE from an Xstation, here are the necessary steps:

- Log in to the Xstation
- type: **export XSTATION=xstation_hostname:0.0**
- type: **export DISPLAY=xstation_hostname:0.0**
- type: **/usr/dt/bin/dtwm**

This all could be put into a shell script. The result is that **dtwm** is called instead of the **mwm**.

Transition Statement — When a user logs in directly to CDE, they need to be aware that their **.profile** file does not get read by default. We will discuss this next.

\$HOME/.dtprofile

- Sets **environment variables** when using **CDE**
- By default, **.profile** will be **ignored**
- To **force a read of .profile**, uncomment last line of **.dtprofile** to read:

```
$ vi $HOME/.dtprofile
...
DTSOURCEPROFILE=true
:wq
```

© Copyright IBM Corporation 2008

Figure 16-18. \$HOME/.dtprofile

AU1310.0

Notes:

Introduction

Your first access to CDE will cause several files and directories to automatically be placed in your **\$HOME** directory. One of these files is called **\$HOME/.dtprofile**.

Your **\$HOME/.dtprofile** file is read each time you log in to the common Desktop Environment (CDE) and is the place to set or override desktop environment variables for your session. Environment variables set in **\$HOME/.dtprofile** are made available to all applications on the desktop. An example of an environment variable that you may want to set in **.dtprofile** is **export ENV=\$HOME/.kshrc** to preserve command recall when using a window within CDE.

Reading .profile

By default, CDE will not read the standard **.profile** file. This can be changed by uncommenting the **DTSOURCEPROFILE** variable assignment at the end of this file.

Problems logging into the system

Errors in **.dtprofile** or **.profile** may prevent a successful login. If, after you log in, your session startup terminates and you are presented with the login screen, this might be the cause.

If this happens, select the **Options->Sessions->Fail-safe Session** item on the login screen, log in and correct the error. The **\$HOME/.dt/startlog** and **\$HOME/.dt/errorlog** files may be helpful in identifying errors.

Instructor Notes:

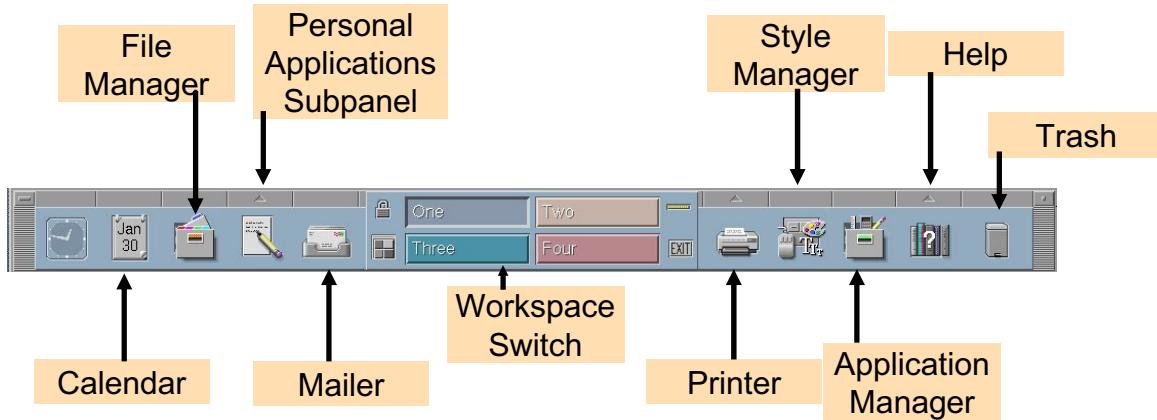
Purpose — Explain the **.dtprofile** file.

Details — If CDE is accessed through the command line using the command **xinit /usr/dt/bin/Xsession**, the **.profile** file will be used instead of **.dtprofile**.

Additional Information — The **.dtprofile** file contains text explaining how to use the file. It mentions that CDE will read **.dtprofile** and **.profile** without an associated terminal emulator such as **dtterm**. For this reason, these scripts must avoid using commands that depend on having an associated terminal emulator or that interact with the user. Details on this and a work around can be found in the comments section of **.dtprofile**.

Transition Statement — Let's take a look at the Front Panel that will be displayed once the user is logged into CDE.

Front Panel



© Copyright IBM Corporation 2008

Figure 16-19. Front Panel

AU1310.0

Notes:

The CDE front panel

The front panel is a window, typically located at the bottom of the screen, which provides a central location for organizing frequently used applications, devices, and information. It exists in all workspaces.

The front panel consists of the main panel, pop-up menus, positioning handles, controls, subpanels and the workspace switches. The front panel is fully customizable.

The controls are pictorial representations of their function. Above are some of the controls as seen on the front panel.

Dual purpose controls

Certain controls, such as the *clock*, are merely indicators reflecting information about your system. Other controls have dual purposes. For example, the *calendar* displays the current date, but it can also be clicked to start a *calendaring application*.

Starting applications

Many of the controls in the front panel start applications when you click them, for example, the **File**, **Style**, and **Application** managers. We will cover these applications in more detail shortly.

Drop zones

Some controls are *drop zones*, for example, the **Printer** and **Trash Can**. It is possible to drag a file from the *File Manager* to one of these controls to print or delete the file respectively.

Application subpanel

Some controls show an arrow above them, for example, the **Help** and the **Personal Applications** controls. Clicking this arrow will display a *subpanel* that can be used with the control.

Instructor Notes:

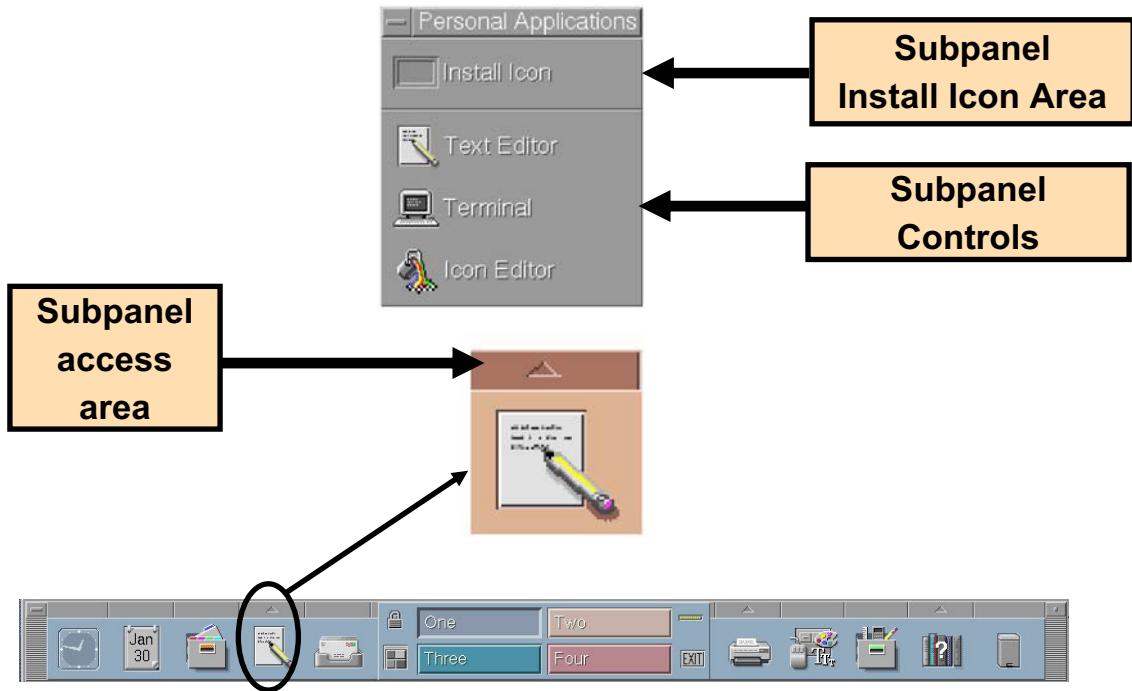
Purpose — Introduce the CDE Front Panel.

Details —

Additional Information —

Transition Statement — Let's introduce subpanels.

Front Panel - Subpanels



© Copyright IBM Corporation 2008

Figure 16-20. Front Panel - Subpanels

AU1310.0

Notes:

Application subpanels

If the control in the front panel has an *arrow button* on top of it, then that control has a subpanel. The example above shows the subpanel of the *Personal Applications* control. Subpanels always contain:

- An *Install Icon* option. Use this option to customize the subpanel.
- A labeled copy of the control in the front panel. In the example above, this is the *Text Editor* option.

Note that this subpanel has a *Terminal* option. This option can be used to bring up a window in which commands can be entered from the command line. The window is called a **dtterm**. This type of window has more function than an **aixterm**. For example, it contains an option bar at the top of the window as well as a scroll bar.

Default subpanel controls

By default, the CDE front panel has three controls that provide subpanels: the Personal Applications control (shown on the visual), the Personal Printers control and the Help control.

Moving subpanels

A subpanel can be moved to another place on the screen and left open for further use. Otherwise, by default, when a control in the subpanel is activated, the subpanel is automatically closed.

Instructor Notes:

Purpose — Explain how some controls have subpanels.

Details — Again, most of the information that needs to be covered is in the student notes.

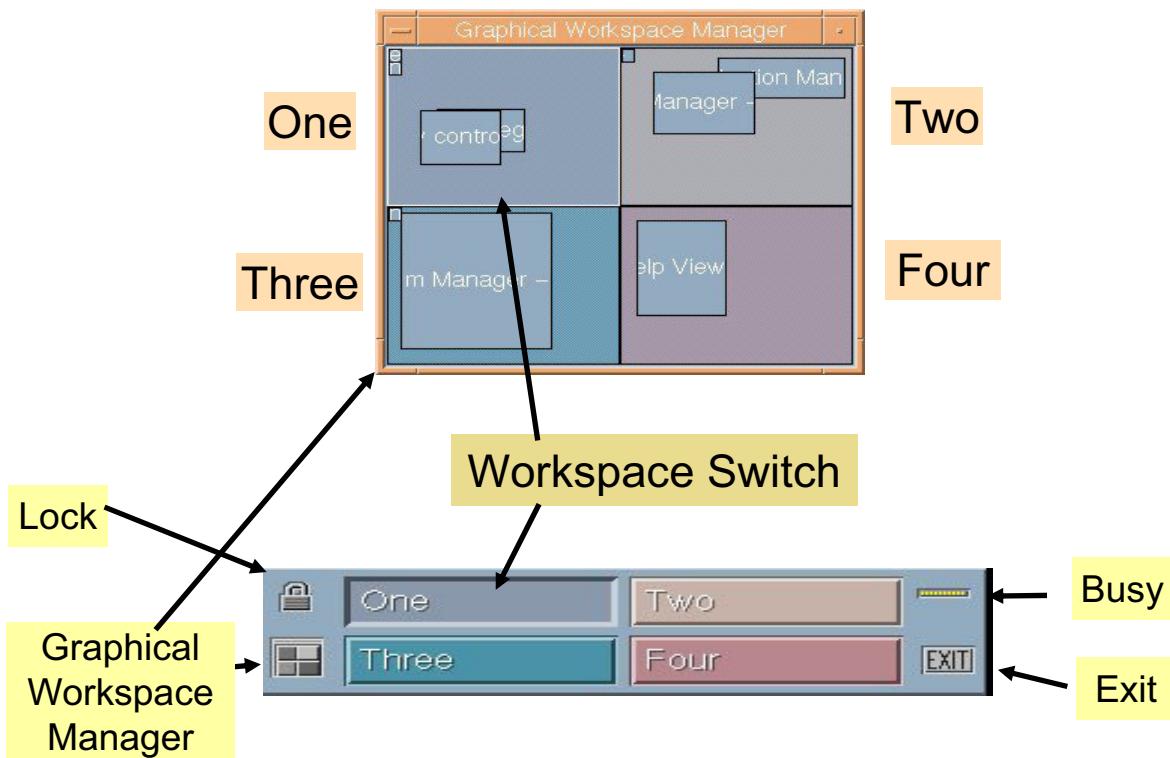
Note that once the subpanel is activated, the arrow above the control points down. Click the down-pointing arrow to close the subpanel.

By default, the CDE front panel has three controls that provide subpanels: The Personal Applications control (shown on the visual), the Personal Printers control and the Help control.

Additional Information —

Transition Statement — The Front Panel also contains a Workspace Switch. Let's discuss that next.

Front Panel - Further Controls



© Copyright IBM Corporation 2008

Figure 16-21. Front Panel - Further Controls

AU1310.0

Notes:

Workspace manager

The Common Desktop Environment provides support for multiple *workspaces*. The *Workspace Switch* is located in the center of the front panel. By default, four workspaces are provided. Click any one of the workspaces to change to another virtual desktop. Workspaces can be added, deleted, or renamed dynamically (more on this in the next unit).

The Workspace Manager area also contains other controls to *lock* the display (unlock by typing the user's password), and *exit* the CDE. The upper right corner of the Workstation Manager shows a *busy* control. The lower left corner of the Workstation Manager can be clicked to access the *Graphical Workspace Manager* window. This window provides a graphical summary of what can be found in each of the workspaces.

Instructor Notes:

Purpose — Explain the Workspace Manager and further front panel controls.

Details — Use the Workspace Manager to access different virtual desktop environments. Users should plan to organize their work so that related tasks appear in the same workspace. This allows them to organize their tasks sensibly, and switch between them easily.

Workspaces can be added or deleted. It is also possible to give them more meaningful names.

The default is four workspaces. The number is actually set by a resource:

Dtwm*workspaceCount:N

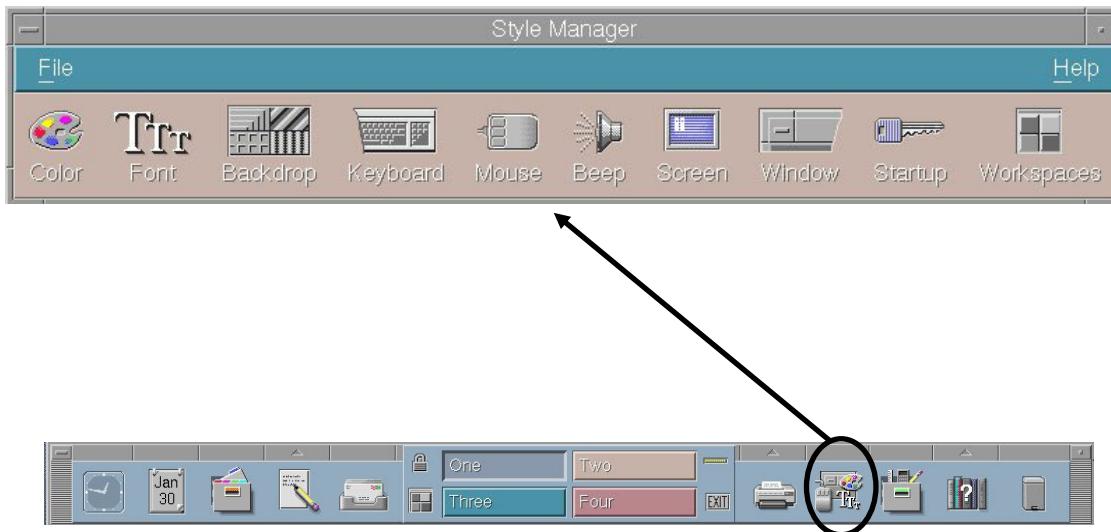
This resource is stored in the file: **\$HOME/.dt/sessions/current**.

Additional Information —

Transition Statement — Let's discuss the Style Manager.

The Style Manager

The **Style Manager** allows interactive **customization** of the desktop environment.



© Copyright IBM Corporation 2008

Figure 16-22. The Style Manager

AU1310.0

Notes:

Customizing the workspace

The Style Manager allows you to customize your workspace colors and palette, application font sizes, workspace backdrop patterns, keyboard volume and character repeat, mouse settings, beep volume, tone and duration, screen saver and screen lock, window focus policies, and how your session begins and ends.

The Style Manager is easily accessible from the front panel control.

Instructor Notes:

Purpose — Introduce the Style Manager.

Details — The various screen savers make an interesting display. Do not go into too much detail, since you will be covering the Style Manager more fully in the next unit.

Additional Information —

Transition Statement — The next control we will discuss is the File Manager.

The File Manager

- Navigate file system
- Work with files and directories
- Drag and drop actions
- Launch applications

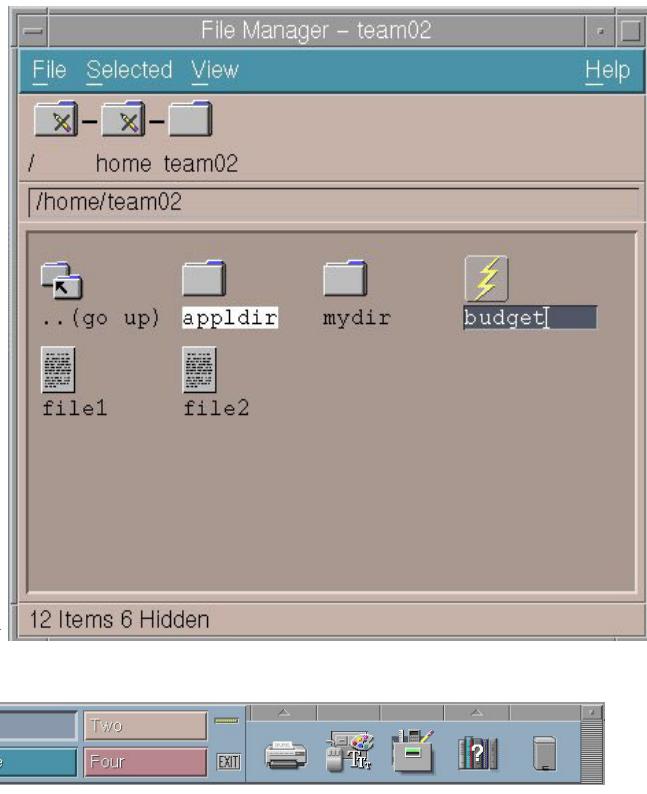


Figure 16-23. The File Manager

AU1310.0

Notes:

Viewing files

Using the File Manager, it is possible to navigate the file structure and view the files and directories in a user friendly way. Invoke the File Manager by clicking its icon on the front panel.

In our example, we are looking at the \$HOME directory for **team02**. Note the graphical representation of the file structure at the top of the window. Both / and **home** show a pencil with a line drawn through it. This means that **team02** does not have write access to these directories. However, it is possible to click any of these directories to view the files within them (assuming the user has read access to the directory).

Note the items shown in **team02**'s \$HOME directory. **appldir** and **mydir** are directories, known as folders to the File Manager. **file1** and **file2** are represented as text files. The file **budget** shows a lightening bolt through it, indicating it is executable.

Instructor Notes:

Purpose — The File Manager is one of the most useful CDE applications. Click the File Manager icon on the Front Panel. The window that opens displays the current directory structure in a graphical format.

Cover the information in the student notes to explain what the different icons within the file manager window mean.

Also point out that at the top of the window is an option bar:

- **File:** This option allows you to create new files and folders (directories), and well as perform a find operation.
- **Selected:** Supports copy, move, Put in Trash (delete), rename, print, and change permissions. This option also allows for a file to be placed in the workspace. This is handy for files that are accessed often.
- **View:** Allows the user to view the folder/directory contents using various selection criteria. This also allows the folder contents to be viewed like a tree structure. Those from the PC world will like this feature!

Additional Information —

Transition Statement — We will next discuss the Application Manager control.

The Application Manager

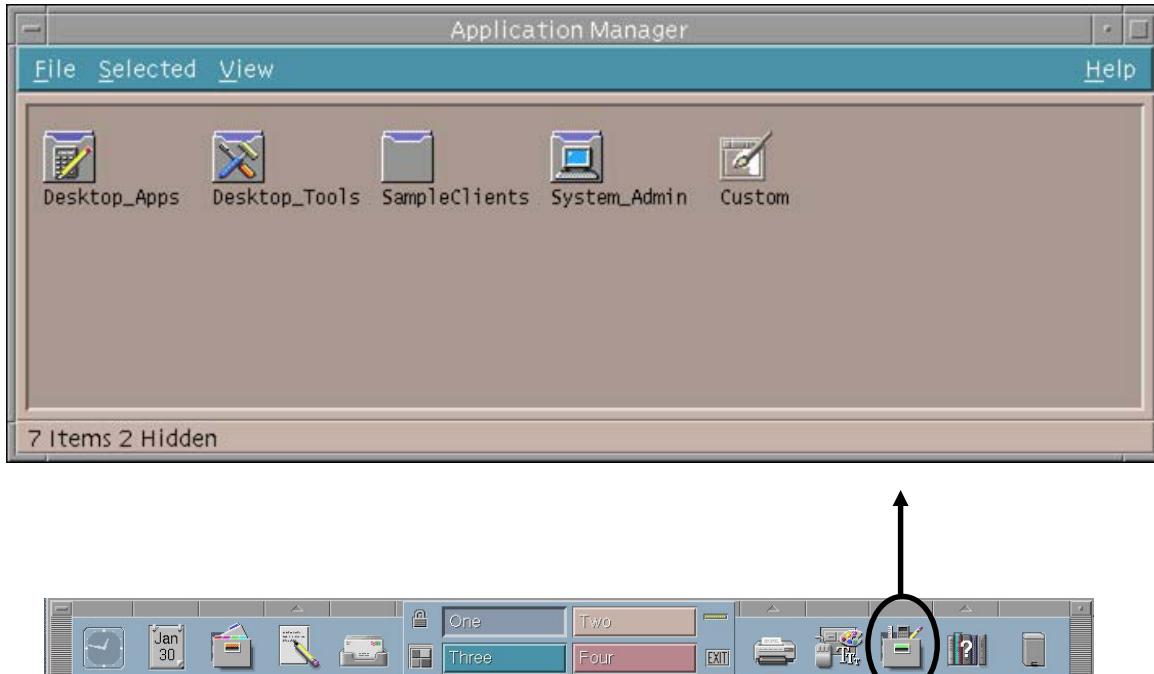


Figure 16-24. The Application Manager

AU1310.0

Notes:

Accessing applications

The Application Manager provides access to applications that are regularly used. The desktop provides built-in applications for tools and utilities that are available with AIX. Several of the built-in applications are really folders containing one or more icons that are used to start applications.

- Desktop_Apps: Provides icons that support functions such as a desktop calculator, a calendar, a man page viewer, an icon editor, the File Manager, the Style Manager, and starting a `dtterm`.
- Desktop_Tools: Provides icons that support functions such as starting an `aixterm`, compressing files, a digital clock, and a spell checker. System management functions such as disk usage reports and system load are also supported.
- System_Admin: Supports system administration functions such as managing users, print queues, and disk space.

Instructor Notes:

Purpose — Introduce the Application Manager.

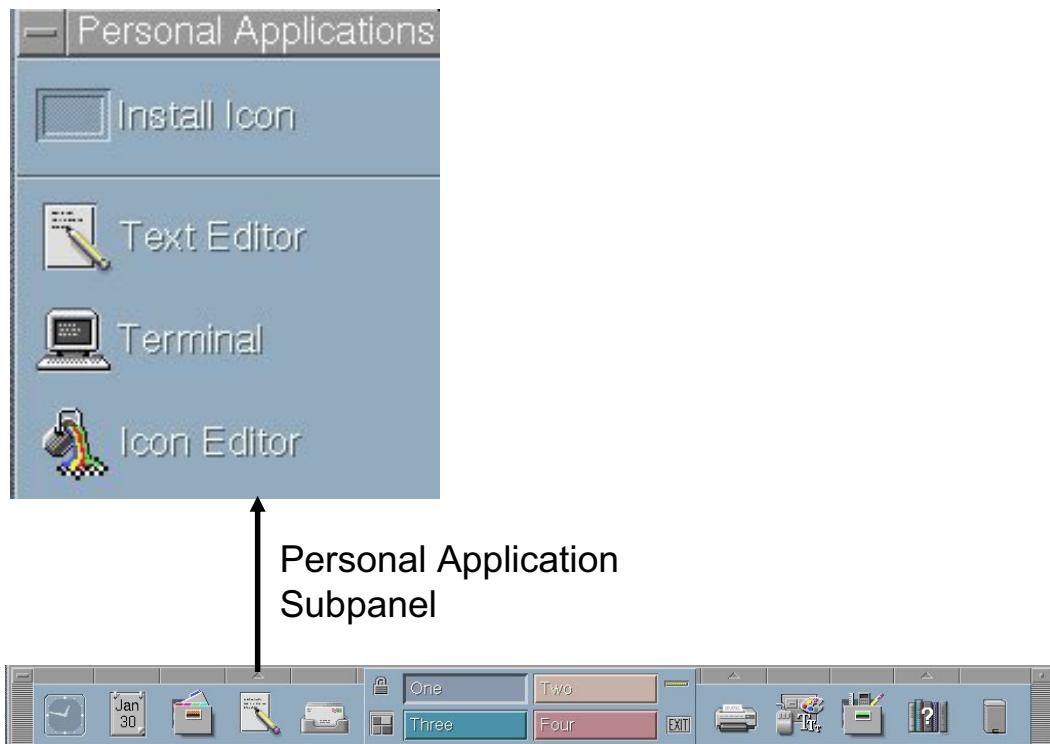
Details — Cover the information in the student notes, but do not go into too much detail. Many of the Application Manager functions are available to the root user only.

If there is an application which you use frequently, it can be placed on the workspace backdrop. Simply drag its icon from the Application Manager and drop it directly on the workspace backdrop. Now it will be available for fast access.

Additional Information —

Transition Statement — The last Front Panel Manager that we will look at is the Personal Applications Manager.

The Personal Applications Manager



© Copyright IBM Corporation 2008

Figure 16-25. The Personal Applications Manager

AU1310.0

Notes:

Personal Applications manager

The Personal Applications manager on the front panel provides a subpanel that can be used to start a terminal (a `dtterm`), run a text editor, or edit icons.

The text editor can be used to create and edit ASCII-based files. Some users may see this as a viable replacement for `vi`.

The terminal emulator creates a `dtterm` which provides more functions than an `aixterm`.

Instructor Notes:

Purpose — Explain the functions of the Personal Applications Manager.

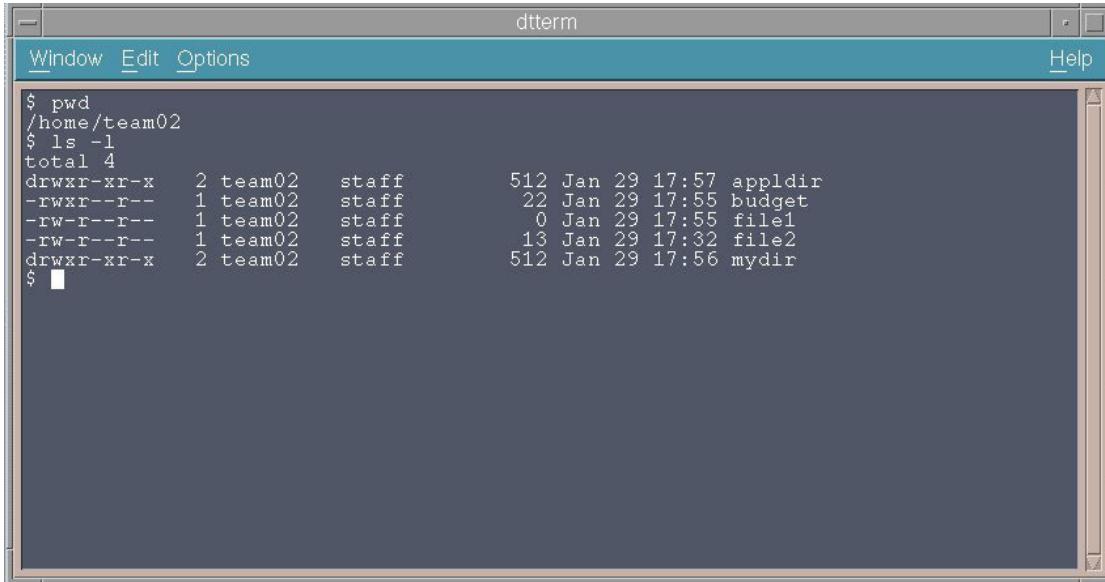
Details — Probably the function that students will use most is the Terminal to obtain a terminal emulator.

In the next unit, we will describe how to place the terminal icon on the Front Panel for Personal Applications (instead of the text editor).

Additional Information —

Transition Statement — Since the students will most likely use the Terminal option quite a bit, let's look at what a `dtterm` looks like.

The Terminal Emulator



© Copyright IBM Corporation 2008

Figure 16-26. The Terminal Emulator

AU1310.0

Notes:

dtterm

The Desktop Terminal Emulator **dtterm** can be used in place of the **aixterm** to enter AIX commands.

The **dtterm** contains a scroll bar as well as a menu bar.

The menu bar options are:

- **Window:** Used to create a new window or close the current window.
- **Edit:** Supports copy and paste functions.
- **Options:** Used to enable/disable the menu bar and the scroll bar. This option also allows you to choose reverse text, a blink rate, font size, and window size.

Instructor Notes:

Purpose — Introduce the `dtterm`.

Details — Students have already worked with the `aixterm`. `dtterm` is the terminal emulator used most in the CDE environment. The Style Manager can be used to alter the colors and characteristics of the `dtterm`.

Additional Information —

Transition Statement — Let's next take a look at the Help available with CDE.

The Help System

- Readily available from:



Help Key



Application Help Menu

Help Subpanel

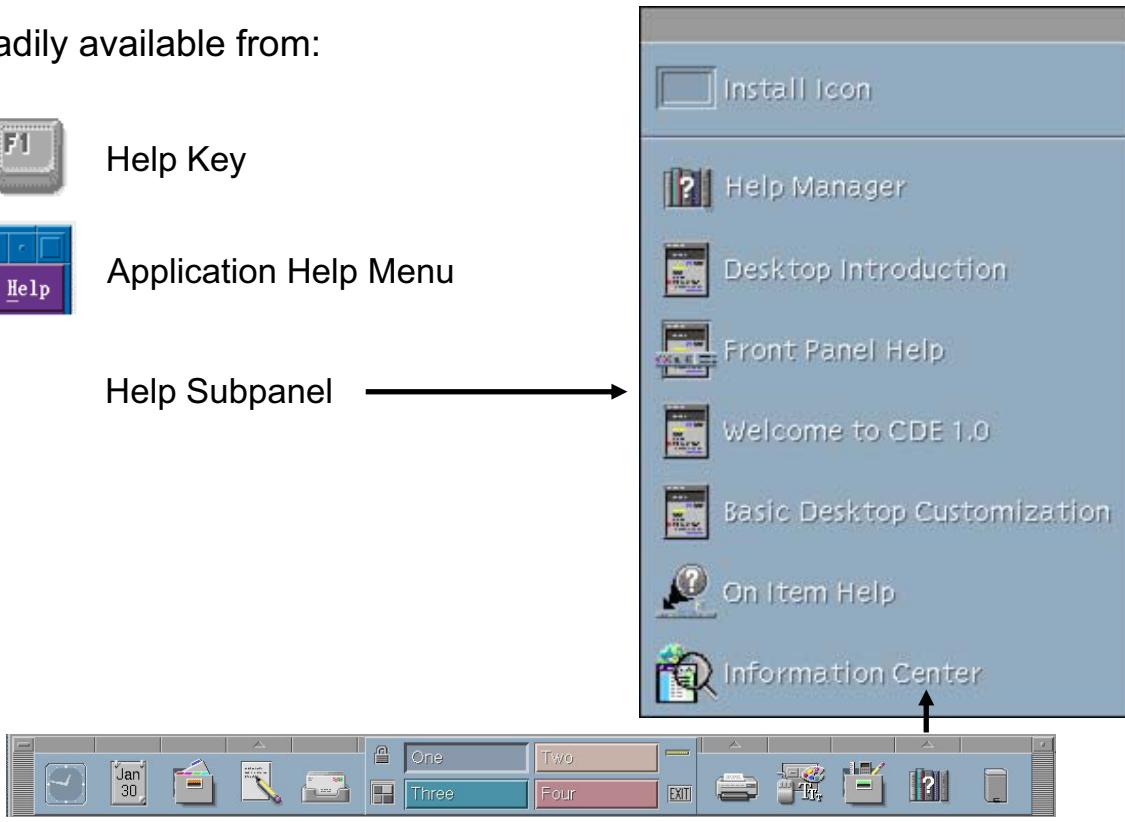


Figure 16-27. The Help System

AU1310.0

Notes:

Using the help system in CDE

Online help is available for each of the standard applications in CDE. Help can be obtained by:

- Pressing **F1** to obtain context help
- Choosing **Help** from any application's pull-down menu
- Clicking the **Help Manager** icon on the front panel

The **Help** subpanel provides several options. Information Center will open the Mozilla Web browser and allow you to access the online documentation.

Instructor Notes:

Purpose — Explain the various ways that help can be obtained in CDE.

Details — Point out the three primary ways to obtain help when using CDE. Point out that the Help subpanel provides yet another way to obtain online documentation.

On the Help subpanel, point out the various self-study helps that are available.

To use On Item Help, click that option, then click the item you would like help on, for example, the mailbox icon on the Front Panel. A window will open providing help in using that function.

The Help Manager subpanel will be different depending on the release level of AIX. This is true for the other subpanels as well.

Additional Information — It is possible to add your own help files to the system.

Transition Statement — That's it, let's go to the Checkpoint Questions.

Checkpoint (1 of 2)

1. If running AIXwindows, what would the following command do? **xclock -bg red -fg white &**

2. List two ways to start a new **aixterm**.

3. Assume two systems: **nysys** (in New York) and **dalsys** (in Dallas). What would be the result if the following command were issued from the AIXwindows environment on dalsys?
rexec nysys xclock -d dalsys:0

4. What is an easy way to customize your AIXwindows environment?

© Copyright IBM Corporation 2008

Figure 16-28. Checkpoint Questions (1 of 2)

AU1310.0

Notes:

Instructor Notes:**Purpose —****Details —**

Checkpoint Solutions (1 of 2)

1. If running AIXwindows, what would the following command do?
`xclock -bg red -fg white &`
This command would start an analog clock with a red background and white foreground.
2. List two ways to start a new `aixterm`.
Choose New Window from the root menu.
Execute the `aixterm` command in the background.
3. Assume two systems: **nysys** (in New York) and **dalsys** (in Dallas). What would be the result if the following command were issued from the AIXwindows environment on dalsys? `rexec nysys xclock -d dalsys:0`
This would display an xclock from nysys on dalsys.
4. What is an easy way to customize your AIXwindows environment?
From the root window, choose custom. This will customize flat files in your home directory for your AIXwindows environment. This will be further discussed in the next unit.

© Copyright IBM Corporation 2008

Additional Information —**Transition Statement —**

Checkpoint (2 of 2)

5. True or false? CDE is designed as a common user interface for the UNIX environment.

6. Match the following terms with their correct meanings:

a) Login Manager	___ Can be used to obtain a dtterm
b) Front Panel	___ Can be used to work with online documentation
c) Personal Applications	___ Used to customize CDE
d) File Manager	___ The application "Launcher"
e) Application Manager	___ Maintains desktop look between sessions
f) Style Manager	___ Provides a GUI to work with files
g) Session Manager	___ Used to manage applications
h) Help Manager	___ Authenticates the user ID

7. True or false? Any environment variables set in .profile will be used by default in the CDE environment.

© Copyright IBM Corporation 2008

Figure 16-29. Checkpoint Questions (2 of 2)

AU1310.0

Notes:

Instructor Notes:**Purpose —****Details —**

Checkpoint Solutions (2 of 2)

5. True or false? CDE is designed as a common user interface for the UNIX environment. **True.**

6. Match the following terms with their correct meanings:

a) Login Manager	<u>c</u> Can be used to obtain a dtterm
b) Front Panel	<u>h</u> Can be used to work with online documentation
c) Personal Applications	<u>f</u> Used to customize CDE
d) File Manager	<u>b</u> The application "Launcher"
e) Application Manager	<u>g</u> Maintains desktop look between sessions
f) Style Manager	<u>d</u> Provides a GUI to work with files
g) Session Manager	<u>e</u> Used to manage applications
h) Help Manager	<u>a</u> Authenticates the user ID

7. True or false? Any environment variables set in .profile will be used by default in the CDE environment. **False. In order for CDE to read .profile, the last line of .dtprofile must be uncommented.**

© Copyright IBM Corporation 2008

Additional Information —**Transition Statement —**

Exercise: Using AIXwindows and CDE



© Copyright IBM Corporation 2008

Figure 16-30. Exercise: Using AIXwindows and CDE

AU1310.0

Notes:

After completing the exercise, you will be able to:

- Start AIXwindows
- Manipulate screen windows using AIXwindows
- Open a new `aixterm` window
- Recognize the various CDE controls on the front panel
- Use the *Help Manager*
- Start a *Terminal* window
- Use the *File Manager*

An optional exercise part shows how you can use AIXwindows in a client/server environment.

Instructor Notes:

Purpose —

Details —

Additional Information —

Transition Statement —

Unit Summary

- AIXwindows is [AIX's windowing system](#). It includes X Windows, Motif, and CDE.
- The [X Client](#) is the application that [displays the graphics](#) while the [X Server](#) controls the display screen and input.
- Start AIXwindows using the [startx](#) command.
- Use the [DISPLAY](#) variable or the [-display](#) option to designate which [server](#) a client will send its output to.
- The CDE provides a [common user interface](#) for the user environment.

© Copyright IBM Corporation 2008

Figure 16-31. Unit Summary

AU1310.0

Notes:

Instructor Notes:

Purpose — To summarize this unit's key points.

Details —

Additional Information —

Transition Statement —

Appendix A. Checkpoint Solutions

Unit 1: Introduction to AIX

Checkpoint Solutions

1. Which part of the operating system interacts directly with the hardware? **Kernel**
2. Which part of the operating system does the user interact with?
 - a) **Shell**
 - b) Kernel
3. Which editor is available across most UNIX platforms? **vi**
4. Write down the names of two AIX graphical user interfaces:
 - a) **AIXwindows**
 - b) **Common Desktop Environment (CDE)**
5. True or false: AIX only supports file systems on hard disks.
False. AIX supports disk file systems, CD-ROM file systems, and network file systems.

© Copyright IBM Corporation 2008

Unit 2: Using the System

Checkpoint Solutions

- What is the correct command syntax in AIX?

```
$ mail newmail -f  
$ mail f newmail  
$ -f mail  
$ mail -f newmail
```

- What command would you use to send mail items? **mail username**
- What are other commands that can be used to communicate with other users? **talk, write, and wall**
- What output would you expect from the following command: **cal 8**?
The calendar for the year 8 AD
- Which command would you use to find out when a particular user logged in?

```
$ who am i  
$ who  
$ finger everyone  
$ finger username
```

© Copyright IBM Corporation 2008

Unit 3: AIX Documentation

Checkpoint Solutions

1. Which command displays manual entries online? **man**
2. Complete the following sentences:
The AIX 6.1 online documentation is loaded on a **document server**. Any other computer in the network with appropriate Web browser software can then become a **document client**.
3. How can you start the documentation from the command line? **infocenter**

© Copyright IBM Corporation 2008

Unit 4: Files and Directories (1 of 2)

Checkpoint Solutions (1 of 2)

1. Using the tree structure shown earlier, and using **/home** as your current directory, how would you refer to the **suba** file in the **pgms** directory using both full and relative path names?

Relative path name: team03/pgms/suba

Full path name: /home/team03/pgms/suba

2. When specifying a path name, what is the difference between the **.** and the **..**?
 - . Specifies current directory
 - .. Specifies parent directory
3. What will the **cd . . / . .** command do?
Change your current working directory two directories higher.
4. What conditions have to be satisfied in order for the **rmdir** command to complete successfully?
 1. **The directory must be empty.**
 2. **You must be at least one directory level higher than the one you are trying to remove.**

© Copyright IBM Corporation 2008

Unit 4: Files and Directories (2 of 2)

Checkpoint Solutions (2 of 2)

5. Match the various options of the ls command with their functions.

-a	<u>-l</u> Provides a long listing of files
-i	<u>-a</u> Lists hidden files
-d	<u>-R</u> Lists subdirectories and their contents recursively
-l	<u>-i</u> Displays the i-node number
-R	<u>-d</u> Displays information about a directory

6. Circle the following valid file names in the following list:

1
aBcDe
-myfile
my_file
my.file
my_file
.myfile

© Copyright IBM Corporation 2008

Unit 5: Using Files

Checkpoint Solutions

1. What is the effect of the following commands?

```
$ cd /home/team01  
$ cp file1 file2
```

The cp command creates a new file, file 2 from a copy of file1. Each copy will have a different name, as shown, file1 and file2. The two copies are independent of each other. If one file is modified, it does not reflect in the second file.

2. What is the effect of the following commands?

```
$ cd /home/team01  
$ mv file1 newfile
```

These commands will rename file1 to newfile. file1 will no longer exist, but instead be shown as newfile.

3. What is the effect of the following commands?

```
$ cd /home/team01  
$ ln newfile myfile
```

The file called newfile is now known as myfile. An ls -l will show both files. An ls -li will show that both files share the same node number. Note that there is still only one physical file on disk. If a change is made to newfile that change will also be reflected if using myfile.

4. List commands that can be used to view the contents of a file.

```
cat, pg, more
```

© Copyright IBM Corporation 2008

Unit 6: File Permissions (1 of 3)

Checkpoint Solutions (1 of 3)

The following questions are for a file called **reporta** which has the following set of permissions: **rwxr-x r-x**

1. What is the mode in octal? **755**
2. Change mode to **rwxr--r--** using symbolic format. **chmod go-x reporta**
3. Repeat the above operation using octal format. **chmod 744 reporta**

Question four is based on the following listing. Assume that the directory **jobs** contains the file **joblog**.

```
$ ls -lR
total 8
drwxr-xr-x  2 judy  finance  512  June  5  11:08  jobs
./jobs:
total 8
-rw-rw-r--  1 judy  finance   100  June  6  12:16  joblog
```

4. Can Fred, who is a member of the finance group, modify the file **joblog**? **Yes, he can, as the file has write permission on the file and has execute permission on the directory.**

© Copyright IBM Corporation 2008

Unit 6: File Permissions (2 of 3)

Checkpoint Solutions (2 of 3)

This question is based on the following listing. Assume that the directory **jobs** contains the directory **work**, which in turn contains the file **joblog**.

```
$ ls -lR
total 8
drwxrwxr-x  3  judy   finance  512  June  5  11:08  jobs

./jobs:
total 8
drwxrwxr-x  2  judy   finance  512  June  5  11:10  work

./jobs/work:
total 8
-rw-rw-r--  1  judy   finance  100  June  6  12:16  joblog
```

5. Can Fred, who is a member of the finance group, modify the file **joblog**? **No, because he does not have execute permission on the intermediate directory, work.**

© Copyright IBM Corporation 2008

Unit 6: File Permissions (3 of 3)

Checkpoint Solutions (3 of 3)

This question is based on the following listing. Assume that the directory **jobs** contains the directory **work**, which in turn contains the file **joblog**.

```
$ ls -lR
total 8
drwxr-xr-x  3 judy  finance  512  June  5  11:08  jobs

./jobs:
total 8
drwxrwxrwx  2 judy  finance  512  June  5  11:10  work

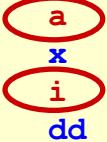
./jobs/work:
total 8
-rw-rw-r--  1 judy  finance   100  June  6  12:16  joblog
```

6. Can Fred, who is a member of the finance group, copy the file **joblog** to his home directory? **Yes**.

© Copyright IBM Corporation 2008

Unit 7: The vi Editor

Checkpoint Solutions

1. When using the `vi` editor, what are the two modes of operation? **text mode and command mode**
2. While using `vi`, how do you get to command mode? **Press the <escape> key. Remember though, the <escape> key is not a toggle. If it is pressed repeatedly, the user remains in command mode.**
3. Which of the following could you use to enter in text?

 - a
 - x
 - i
 - dd
4. While in command mode, pressing the u key repeatedly will "undo" all previously entered commands. True or False? **False. The u command will only undo the previous command.**
5. `vi` can be used to globally change the first occurrence of a pattern on every line with a given pattern. True or False? **True.**

© Copyright IBM Corporation 2008

Unit 8: Shell Basics (1 of 2)

Checkpoint Solutions (1 of 2)

1. What will the following command match?

```
$ ls ???[!a-z]*[0-9]t
```

This will list all the files that match the following criteria:

- the first three characters can be anything
- the fourth character must not be from the range a to z
- zero or more characters can follow
- the second-last character must be from the range 0 to 9
- the last character must be a t.

2. For questions 2-4, indicate where the standard input, standard output and standard error will go.

\$ cat file1	
standard input (0):	keyboard
standard output (1):	screen
standard error (2):	screen

3. \$ mail tim < letter

standard input (0):	letter
standard output (1):	screen
standard error (2):	screen

© Copyright IBM Corporation 2008

Unit 8: Shell Basics (2 of 2)

Checkpoint Solutions (2 of 2)

4. `$ cat .profile > newprofile 2>1`
standard input (0): keyboard
standard output (1): newprofile
standard error (2): a file named 1

For questions 5, 6, and 7, create command lines to display the content of **filea** using **cat** and then perform the following:

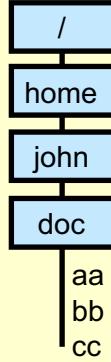
5. Place the output of the command in **fileb** and the errors in **filec**.
`$ cat filea > fileb 2> filec`
6. Place the output of the command in **fileb** and associate any errors with the output in **fileb**.
`$ cat filea > fileb 2>&1`
7. Place the output in **fileb** and discard any error messages. (Do not display or store error messages.)
`$ cat filea > fileb 2> /dev/null`

© Copyright IBM Corporation 2008

Unit 9: Using Shell Variables (1 of 2)

1. What are the results of the following commands? (Assume: the home directory is **/home/john**, the current directory is **/home/john/doc**, and it contains files **aa**, **bb** and **cc**.)

```
$ pwd  
/home/john/doc
```



2. \$ echo "Home directory is \$HOME"

Home directory is /home/john

3. \$ echo 'Home directory is \$HOME'

Home directory in \$HOME

© Copyright IBM Corporation 2008

Unit 9: Using Shell Variables (2 of 2)

4. \$ echo "Current directory is `pwd`"
Current directory is /home/john/doc

5. \$ echo "Current directory is \$(pwd)"
Current directory is /home/john/doc

6. \$ echo "Files in this directory are *"
File in this directory are *

7. \$ echo * \$HOME
aa bb cc /home/john

8. \$ echo *

© Copyright IBM Corporation 2008

Unit 10: Processes

Checkpoint Solutions

1. When would you execute a shell script using the dot (.) notation? Why? **When you are using the script to change variable values in the current shell.**
2. What is the command that is used to carry down the value of a variable into the subshell? **export variable_name**
3. What would be the value of x at the end of the following steps?

```
$ ( ... login shell ... )  
$ ksh  
$ x=50  
$ export x  
$ <ctrl -d>  
$ (what is the value of x set to now?)
```

x would have the value it had before starting the subshell. If the login shell had not set the variable, then after return from the subshell it would still not be set.

© Copyright IBM Corporation 2008

Unit 11: Controlling Processes

Checkpoint Solutions

1. What option would you use with the **ps** command to show the detailed commands that you are running? **ps -f**
2. True or false? As an ordinary user, you can only kill your own jobs and not those of other users. **True**
3. Which is the strongest signal that can be sent to a process to terminate it? **signal 9**
4. It is always sensible to start long jobs in the background with the **nohup** command. Why is this? **The job will not lock up the user's terminal, and will continue to run when you log off the system.**
5. What is the name for special never-ending system processes in the UNIX environment? **Daemons**

© Copyright IBM Corporation 2008

Unit 12: Customizing the User Environment

Checkpoint Solutions

1. Which file would you use to customize your user environment? Why? **\$HOME/.profile as this is the file that overrides the /etc/profile which is the system-defined file.**
2. What do the following variables define on your system?

PS1: primary prompt string (that is, your prompt)

TERM: the terminal type

PATH: the path of directories that is searched, in order to locate an executable

© Copyright IBM Corporation 2008

Unit 13: AIX Utilities

Checkpoint Solutions

1. Which commands would you use to locate all the files in your system that began with the string "smit"?

`find / -name 'smit*'`

2. What is the following command doing?

`$ ps -ef | grep -w root | grep -w netscape`

List all processes which have both the root and netscape strings on their ps -ef report lines.

3. Indicate what the following command is doing:

`$ ls -l /home | egrep 'txt$ | team01$' | sort -r +7 | tail +4 | head -5`

A long listing will be carried out from the /home directory, and lines ending with txt or team01 will be picked out and piped through and sorted with the following results:

Once the sort is completed, the output will be piped to tail which will only write line 4 and beyond through the pipe to the head command which will only write to the screen the first 5 lines that it receives (lines 4 through 8 of the sorted file).

© Copyright IBM Corporation 2008

Unit 14: AIX Utilities, Part II

Checkpoint Solutions

1. True or false? `find`'s most important characteristic is its ability to travel up through the file tree hierarchy. **False.**
2. True or false? When quoted metacharacters are used with `find`, the shell will first expand the wildcard then pass control to `find`. **False.**
3. Which command is used to determine the type of data in a file?
`cmp`
`diff`
file
`diffcmp`
4. True or false? `diff` compares text files only. **True.**
5. True or false? The `compress` command will delete the file it is compressing and replace it with the compressed file also renaming it with a `.z` extension. **False. The extension is an uppercase .Z**
6. To display non-printable characters in a file or directory, use:
`ls -l`
cat -vte
`diff -c`
`cmp`

© Copyright IBM Corporation 2008

Unit 15: Additional Shell Features

Checkpoint Solutions

- What will the following piece of code do?

```
TERMTYPE=$TERM
if [ $TERMTYPE != "" ]
then
if [ -f /home/team01/customized_script ]
then
/home/team01/customized_script
else
echo No customized script available !
fi
else
echo You do not have a TERM variable set !
fi
```

The script will set a variable **TERMTYPE** to the value of the **TERM** variable. In the if statement the **TERMTYPE** variable will be tested to see if it is not empty. If it is not, then a second check will be carried out to ensure that the **/home/team01/customized_script** file is an ordinary file. If it is then it will be executed.

(For our example we will assume that this file contains some extra customized features.) If this file is not an ordinary file, then a message will be sent to the user stating this. If the initial test fails, that is, the **TERMTYPE** variable is empty, then again a message will be sent to the user.

- Write a script which will accept two arguments, multiply them together, and display the result. **expr \$1 * \$2**

© Copyright IBM Corporation 2008

Unit 16: The AIX Graphical User Interface

Checkpoint Solutions (1 of 2)

1. If running AIXwindows, what would the following command do?
`xclock -bg red -fg white &`
This command would start an analog clock with a red background and white foreground.
2. List two ways to start a new `aixterm`.
**Choose New Window from the root menu.
Execute the `aixterm` command in the background.**
3. Assume two systems: **nysys** (in New York) and **dalsys** (in Dallas). What would be the result if the following command were issued from the AIXwindows environment on dalsys? `rexec nysys xclock -d dalsys:0`
This would display an xclock from nysys on dalsys.
4. What is an easy way to customize your AIXwindows environment?
From the root window, choose custom. This will customize flat files in your home directory for your AIXwindows environment. This will be further discussed in the next unit.

© Copyright IBM Corporation 2008

Unit 16: The AIX Graphical User Interface

Checkpoint Solutions (2 of 2)

5. True or false? CDE is designed as a common user interface for the UNIX environment. **True.**

6. Match the following terms with their correct meanings:

a) Login Manager	<u>c</u> Can be used to obtain a dtterm
b) Front Panel	<u>h</u> Can be used to work with online documentation
c) Personal Applications	<u>f</u> Used to customize CDE
d) File Manager	<u>b</u> The application "Launcher"
e) Application Manager	<u>g</u> Maintains desktop look between sessions
f) Style Manager	<u>d</u> Provides a GUI to work with files
g) Session Manager	<u>e</u> Used to manage applications
h) Help Manager	<u>a</u> Authenticates the user ID

7. True or false? Any environment variables set in .profile will be used by default in the CDE environment. **False. In order for CDE to read .profile, the last line of .dtprofile must be uncommented.**

© Copyright IBM Corporation 2008

Appendix C: Customizing AIXWindows

Checkpoint Solutions

1. Match the AIXwindows startup file with its function:

a) .xinitrc	b Sets default characteristics for AIXwindows resources
a) .Xdefaults	a Starts the Motif window manager
b) .mwmrc	c Defines the function of the root menu and the window menu
2. Name two ways the **.Xdefaults** file can be customized. **Edit manually, use the AIXwindows custom tool.**
3. True or false? The AIXwindows custom tool saves all customization choices in the **.xinitrc** file. **FALSE, most changes are stored in the .Xdefaults file.**
4. What command is used to change the appearance of the root window? **The xsetroot command.**
5. Where would the **xsetroot** command be placed to make a permanent change to the root window? **In .xinitrc.**

© Copyright IBM Corporation 2008

Appendix D: CDE User Customization

Checkpoint Solutions

1. How do you customize the screen saver in your desktop environment ?
By using the Style Manager

2. True or false: You can have more than four workspaces on the CDE front panel.
True

3. Describe how controls can be added to the CDE front panel.
First add the control to a subpanel. Copy the definition file and anchor the control in the front panel.

© Copyright IBM Corporation 2008

Appendix B. Command Summary

Startup, Logoff and Shutdown

<Ctrl+d> (exit)	log off the system (or the current shell).
shutdown	shuts down the system by disabling all processes. If in single-user mode, may want to use -F option for fast shutdown. -r option will reboot system. Requires user to be root.

Directories

mkdir	make directory
cd	change directory. Default is \$HOME directory.
rmdir	remove a directory (beware of files starting with ".")
rm	remove file; -r option removes directory and all files and subdirectories recursively.
pwd	print working directory
ls	list files -a (all) -l (long) -d (directory information) -r (reverse alphabetic) -t (sort by time changed) -c (multi column format) -R (recursive listing) -F (places / after each directory name & * after each exec file)

Files - Basic

cat	list file contents (concatenate). Can open a new file with redirection, for example, cat > newfile. Use <Ctrl>d to end input.
chmod	change permission mode for files or directories. <ul style="list-style-type: none"> • chmod =+- files or directories • (r,w,x = permissions and u, g, o, a = who) • can use + or - to grant or revoke specific permissions. • can also use numerical, 4 = read, 2 = write, 1 = execute. • can sum them, first is user, next is group, last is other. • for example, "chmod 746 file1" is user = rwx, group = r, other = rw.
chown	change owner of a file

chgrp	change group of a files
cp	copy file
del	delete files with prompting (rm for no prompting)
mv	move and rename file
pg	list file contents by screen (page) h (help) q (quit) <cr> (next pg) f (skip 1 page) l (next line) d (next 1/2 page) \$ (last page) p (previous file) n (next file) . (redisplay current page) /string (find string forward) ?string (find string backward) -<#> (move backward <#> pages) +<#> (move forward <#> pages)
rm	remove (delete) files (-r option removes directory and all files and subdirectories)
head	print first several lines of a file
tail	print last several lines of a file
wc	report the number of lines (-l), words (-w), characters (-c) in a file. No options gives lines, words, and characters.
su	switch user
id	displays your user ID environment how it is currently set
tty	displays the device that is currently active. Very useful for Xwindows where there are several pts devices that can be created. It is nice to know which one you have active. who am i will do the same.

Files - Advanced

awk	programmable text editor
banner	display banner
cal	calendar (cal [month] year)
cut	cut out specific fields from each line of a file
diff	differences between two files

find	find files anywhere on disk. Specify location by path (will search all subdirectories under specified directory). -name f1 (file names matching f1 criteria) -user u1 (files owned by user u1) -size +n (or -n) (files larger (or smaller) than n blocks) -mtime +x (-x) (files modified more (less) than x days ago) -perm num (files whose access permissions match num) -exec (execute a command with results of find command) -ok (execute a command interactively with results of find command) -o (logical or) -print (display results, default) find syntax: find path expression action for example, find / -name "*.txt" -print or find / -name "*.txt" -exec li -l {} \; executes li -l where names found are substituted for {} ; indicates end of command to be executed and \ removes usual interpretation as command continuation character
grep	search for pattern, for example, grep pattern files . Pattern can include regular expressions. -c (count lines with matches, but do not list) -l (list files with matches, but do not list) -n (list line numbers with lines) -v (find files without pattern) expression metacharacters <ul style="list-style-type: none"> • [] matches any one character inside. • with a - in [] will match a range of characters. • &and. matches BOL when &and. begins the pattern. • \$ matches EOL when \$ ends the pattern. • . matches any single character. (same as ? in shell). • * matches 0 or more occurrences of preceding character. (Note: ".*" is the same as "*" in the shell).
sed	stream (text) editor. Used with editing flat files.
sort	sort and merge files -r (reverse order); -u (keep only unique lines)

Editors

ed	line editor
vi	screen editor
INed	LPP editor

emacs screen editor +

Shells, Redirection and Pipelining

< (read)	redirect standard input; for example, command < file reads input for command from file.
> (write)	redirect standard output; for example, command > file writes output for command to file overwriting contents of file.
>> (append)	redirect standard output; for example, command >> file appends output for command to the end of file.
2>	redirect standard error (to append standard error to a file, use command 2>> file) combined redirection examples: command < infile > outfile 2> errfile command >> appendfile 2>> errfile < infile
;	command terminator used to string commands on single line
 	pipe information from one command to the next command. For example, ls cpio -o > /dev/fd0 will pass the results of the ls command to the cpio command.
\	continuation character to continue command on a new line. Will be prompted with > for command continuation.
tee	reads standard input and sends standard output to both standard output and a file. For example, ls tee ls.save sort results in ls output going to ls.save and piped to sort command.

Metacharacters

*	any number of characters (0 or more)
?	any single character
[abc]	[] any character from the list
[a-c]	[] match any character from the list range
!	not any of the following characters (for example, leftbox !abc right box)
;	command terminator used to string commands on a single line
&	command preceding and to be run in background mode
#	comment character
\	removes special meaning (no interpretation) of the following character removes special meaning (no interpretation) of character in quotes

"	interprets only \$, back quote, and \ characters between the quotes.
'	used to set variable to results of a command; for example, <code>now='date'</code> sets the value of <code>now</code> to current results of the <code>date</code> command.
\$	preceding variable name indicates the value of the variable.

Variables

=	set a variable (for example, <code>d=day</code> sets the value of <code>d</code> to <code>day</code>). Can also set the variable to the results of a command by the ' character; for example, <code>now=date</code> sets the value of <code>now</code> to the current result of the <code>date</code> command.
HOME	home directory
PATH	path to be checked
SHELL	shell to be used
TERM	terminal being used
PS1	primary prompt characters, usually \$ or #
PS2	secondary prompt characters, usually >
\$?	return code of the last command executed
set	displays current local variable settings
export	exports variable so that they are inherited by child processes
env	displays inherited variables
echo	echo a message (for example, <code>echo HI</code> or <code>echo \$d</code>). Can turn off carriage returns with \c at the end of the message. Can print a blank line with \n at the end of the message.

Tapes and Diskettes

<code>format</code>	AIX command to format a diskette
<code>backup</code>	backs up individual files. -i reads file names from standard input -v list files as backed up; for example, <code>backup -iv -f/dev/rmt0 file1, file2</code> -u backup file system at specified level; for example, <code>backup -level -u filesystem</code>

	Can pipe list of files to be backed up into command; for example, <code>find . -print backup -ivf/dev/rmt0</code> where you are in directory to be backed up.
<code>restore</code>	restores commands from backup -x restores files created with backup -i -v list files as restore -T list files stored of tape or diskette -r restores filesystem created with <code>backup -level -u</code> ; for example, <code>restore -xv -f/dev/rmt0</code>
<code>cpio</code>	copies to and from an I/O device. Destroys all data previously on tape or diskette. For input, must be able to place files in the same relative (or absolute) path name as when copied out (can determine path names with -it option). For input, if file exists, compares last modification date and keeps most recent (can override with -u option). -o (output) -i (input) -t (table of contents) -v (verbose) -d (create needed directory for relative path names) -u (unconditional to override last modification date) for example, <code>cpio -o > /dev/fd0</code> <code>file1</code> <code>file2</code> <code><Ctrl+d></code> or <code>cpio -iv file1 < /dev/fd0</code>
<code>tar</code>	alternative utility to back up and restore files
<code>pax</code>	alternative utility to cpio and tar commands

Transmitting

<code>mail</code>	send and receive mail. With userid sends mail to userid. Without userid, displays your mail. When processing your mail, at the ? prompt for each mail item, you can: <code>d</code> - delete <code>s</code> - append <code>q</code> - quit <code>Enter</code> - skip <code>m</code> - forward
<code>mailx</code>	upgrade of mail

uucp	copy file to other UNIX systems (UNIX to UNIX copy)
uuto/uupick	send and retrieve files to public directory
uux	execute on remote system (UNIX to UNIX execute)

System Administration

df	display filesystem usage
installp	install program
kill <pid>	kill batch process with PID (find using ps); kill -9 <pid> will absolutely kill process
mount	associate logical volume to a directory; for example, mount device directory
ps -ef	shows process status
umount	disassociate filesystem from directory
smit	system management interface tool

Miscellaneous

banner	displays banner
date	displays current date and time
newgrp	change active groups
nice	assigns lower priority to following command (for example, nice ps -f)
passwd	modifies current password
sleep n	sleep for n seconds
stty	show and or set terminal settings
touch	create a zero length files
xinit	initiate X Windows
wall	sends message to all logged-in users
who	list users currently logged in (who am i identifies this user)
man	displays manual pages

System Files

/etc/group	list of groups
/etc/motd	message of the day, displayed at login.

/etc/passwd	list of users and sign-on information. Password shown as !. Can prevent password checking by editing to remove !.
/etc/profile	system-wide user profile executed at login. Can override variables by resetting in the user's .profile file.

Shell Programming Summary

Variables

var=string	set variable to equal string. (NO SPACES). Spaces must be enclosed by double quotes. Special characters in string must be enclosed by single quotes to prevent substitution. Piping (!), redirection (<, >, >>), and & symbols are not interpreted.
\$var	gives value of var in a compound
echo	displays value of var; for example, echo \$var
HOME	= home directory of user
MAIL	= mail file name
PS1	= primary prompt characters, usually \$ or #
PS2	= secondary prompt characters, usually >
PATH	= search path
TERM	= terminal type being used
export	exports variables to the environment
env	displays environment variables settings
 \${var:-string}	gives value of var in a command. If var is null, uses string instead.
\$1 \$2 \$3...	positional parameters for variable passed into the shell script
\$*	used for all arguments passed into shell script
\$#	number of arguments passed into shell script
\$0	name of shell script
\$\$	process ID <pid>
\$?	last return code from a command

Commands

#	comment designator
&&	logical-and. Run command following && only if command preceding && succeeds (return code = 0).
 	logical-or. Run command following only if command preceding fails (return code < > 0).
exit n	used to pass return code n from shell script. Passed as variable \$? to parent shell
expr	arithmetic expressions Syntax: expr expression1 operator expression2 operators: + - * (multiply) / (divide) % (remainder)
for loop	for n (or: for variable in \$*); for example: do command done
if-then-else	if test expression then command elif test expression then command else then command fi
read	read from standard input
shift	shifts arguments 1-9 one position to the left and decrements number of arguments
test	used for conditional test, has two formats. if test expression (for example, if test \$- -eq 2) if [expression] (for example, if [\$# -eq 2]) (spaces req'd) integer operators: -eq (=) -lt (<) -le (=<) -ne (<>) -gt (>) -ge (=>) string operators: = (equal) != (not eq.) -z (zero length)

- file status (for example, -opt file1)
- f (ordinary file)
- r (readable by this process)
- w (writable by this process)
- x (executable by this process)
- s (non-zero length)

```
while loop      while test expression  
                  do  
                  command  
                  done
```

Miscellaneous

sh execute shell script in the sh shell -x (execute step by step - used for debugging shell scripts)

vi Editor

Entering vi

vi file	edits the file named file
vi file file2	edit files consecutively (via :n)
.exrc	file that contains the vi profile
wm=nn	sets wrap margin to nn Can enter a file other than at first line by adding + (last line), +n (line n), or +/pattern (first occurrence of pattern).
vi -r	lists saved files
vi -r	file recover file named file from crash
:n	next file in stack
:set all	show all options
:set nu	display line numbers (off when set nonu)
:set list	display control characters in file
:set wm=n	set wrap margin to n
:set showmode	set display of INPUT when in input mode

Read, Write, Exit

:w	write buffer contents
:w file2	file2 write buffer contents to file2
:w >> file2	write buffer contents to end of file2
:q	quit editing session
:q!	quit editing session and discard any changes
:r file2	read file2 contents into buffer following current cursor
:r! com	read results of shell command com following current cursor
:!	exit shell command (filter through command)
:wq or ZZ	write and quit edit session

Units of Measure

h, l	character left, character right
k or <Ctrl+p>	move cursor to character above cursor
j or <Ctrl+n>	move cursor to character below cursor
w, b	word right, word left

&hat., \$	beginning, end of current line
<CR> or +	beginning of next line
-	beginning of previous line
G	last line of buffer

Cursor Movements

Can precede cursor movement commands (including cursor arrow) with number of times to repeat; for example, 9--> moves right 9 characters.

0	move to first character in line
\$	move to last character in line
&and.	move to first nonblank character in line
fx	move right to character x
Fx	move left to character x
tx	move right to character preceding character x
Tx	move left to character preceding character x
;	find next occurrence of x in same direction
,	find next occurrence of x in opposite direction
w	tab word (nw = n tab word) (punctuation is a word)
W	tab word (nw = n tab word) (ignore punctuation)
b	backtab word (punctuation is a word)
B	backtab word (ignore punctuation)
e	tab to ending char. of next word (punctuation is a word)
E	tab to ending char. of next word (ignore punctuation)
(move to beginning of current sentence
)	move to beginning of next sentence
{	move to beginning of current paragraph
}	move to beginning of next paragraph
H	move to first line on screen
M	move to middle line on screen
L	move to last line on screen
<Ctrl+f>	scroll forward 1 screen (3 lines overlap)
<Ctrl+d>	scroll forward 1/2 screen
<Ctrl+b>	scroll backward 1 screen (0 line overlap)

<Ctrl+u>	scroll backward 1/2 screen
G	go to last line in file
nG	go to line n
<Ctrl+g>	display current line number

Search and Replace

/pattern	search forward for pattern
?pattern	search backward for pattern
n	repeat find in the same direction
N	repeat find in the opposite direction

Adding Text

a	add text after the cursor (end with <esc>)
A	add text at end of current line (end with <esc>)
i	add text before the cursor (end with <esc>)
I	add text before first nonblank char in current line
o	add line following current line
O	add line before current line
<esc>	return to command mode

Deleting Text

<Ctrl+w>	undo entry of current word
@	kill the insert on this line
x	delete current character
dw	delete to end of current word (observe punctuation)
diw	delete to end of current word (ignore punctuation)
dd	delete current line
d	erase to end of line (same as d\$)
d)	delete current sentence
d}	delete current paragraph
dg	delete current line through end of buffer
d&and.	delete to the beginning of line
u	undo last change command

U restore current line to original state before modification

Replacing Text

ra replace current character with a
R replace all characters overtyped until <esc> is entered
s delete current character and append test until <esc>
s/s1/s2 replace s1 with s2 (in the same line only)
S delete all characters in the line and append text
cc replace all characters in the line (same as S)
ncx delete n text objects and enter append mode
C replace all characters from cursor to end of line.

Moving Text

p paste last text deleted after cursor (xp will transpose 2 characters)
P paste last text deleted before cursor
nyx yank n text objects of type x w, b = words,) = sentences, } = paragraphs, \$ = end-of-line, and no x indicates lines. Can then paste them with "p" command. Yank does not delete the original.
"ayy can use named registers for moving, copying, cut/paste with "ayy" for register a (use registers a-z). Can then paste them with "ap" command.

Miscellaneous

. repeat last command
J join current line w/next line

Appendix C. Customizing AIXwindows

Estimated Time

00:50

What This Unit Is About

This unit provides basic information on how a user can customize their AIXwindows environment.

What You Should Be Able to Do

After completing this unit, you should be able to:

- Explain the purpose of the AIXwindows startup files: `.xinitrc`, `.Xdefaults`, and `.mwmrc`
- Use the AIXwindows *custom* tool to customize a user's AIXwindows environment
- Use the `xsetroot` command to customize the root window

How You Will Check Your Progress

Accountability:

- Checkpoint questions
- Exercise

Unit Objectives

After completing this unit, you should be able to:

- Explain the purpose of the AIXwindows startup files: **.xinitrc**, **.Xdefaults**, **.mwmrc**
- Use the AIXwindows custom tool to customize a user's AIXwindow's environment
- Use the xsetroot command to customize the root window

© Copyright IBM Corporation 2008

Figure C-1. Unit Objectives

AU1310.0

Notes:

Instructor Notes:

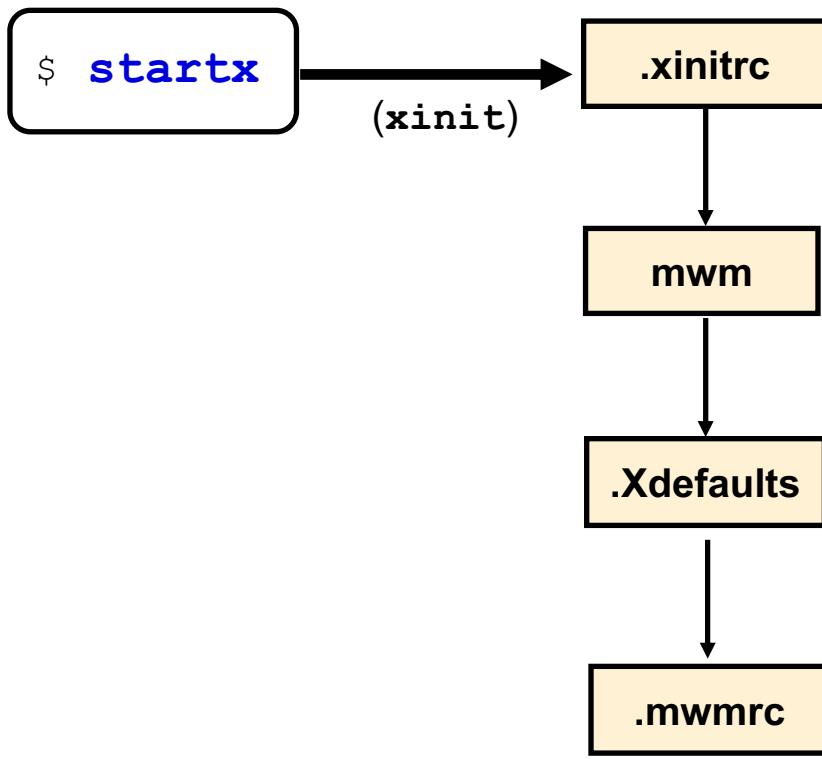
Purpose — To introduce this unit's objectives to the students.

Details —

Additional Information —

Transition Statement — Let's start with an overview of how AIXwindows starts up.

AIXwindows Startup Overview



© Copyright IBM Corporation 2008

Figure C-2. AIXwindows Startup Overview

AU1310.0

Notes:

Introduction

As you have already learned, execute the **startx** shell script to start the AIXwindows environment. If you have an LFT attached directly to an IBM System p system, the **startx** shell script will execute the **xinit** command. In this environment, running the **xinit** command will produce the same result. On an X Station, you must use the **startx** command as **xinit** will not work.

AIXwindows startup

What happens next may vary, depending on your environment. In general though, the following events will occur:

- A customizable shell script called **.xinitrc** will execute. This file will start a user's clients (such as an **aixterm** and the **xclock**) and will then start the Motif window

manager (**mwm**). A user may use the system-wide version of **xinitrc**, or use their own customized version stored in their \$HOME directory.

- The **mwm** program will start Motif and tailor it according to information in two files: **.Xdefaults** and **.mwmrc**.
- The **.Xdefaults** file contains a user's personal preferences for visual characteristics such as colors, fonts, focus policy, and use of scroll bars. This file is located in the user's \$HOME directory and is optional.
- The **.mwmrc** file is used to customize such things as the Root menu, the Window menu and the behavior of the mouse. Like the **.xinitrc** file, a user may use the system-wide version of the file, or use their own customized version stored in their \$HOME directory.

Instructor Notes:

Purpose — Provide the big picture in terms of what happens when AIXwindows is started.

Details — Cover the flow of AIXwindows as shown in the visual and explained in the student notes. Detail on each of the files will be coming up shortly. Be sure to explain that if these files do not exist in a user's \$HOME directory, then the system-wide version of the file will be used. If the file does exist in the user's \$HOME directory, it will override the system-wide defaults.

The ability for a user to own their own AIXwindows startup files allows them to start a windows environment according to their personal preferences. For example, a user may wish to start three `aixterm` windows upon AIXwindows startup. Tailoring of these files will support this customized environment.

Do not cover the files listed in detail as they will be covered in more detail shortly.

Additional Information —

Transition Statement — Let's look at each file in more detail next. We will first discuss the `.xinitrc` file.

.xinitrc

```
*****
# start xclock then sleep 1 to make sure it can get started.
*****
xclock -geometry -0+0 -fg AntiqueWhite1 -bg grey60 -update 1 &
sleep 1
...
*****
```

xclock


```
*****
#
# Start the X clients. Change the following lines to
# whatever command(s) you desire!
# The default clients are an analog clock (xclock), a
# terminal emulator (aixterm), and the Motif window
# manager (mwm).
#
*****
```

Background color

aixterm

Motif Window Manager

© Copyright IBM Corporation 2008

Figure C-3. .xinitrc

AU1310.0

Notes:

System or user .xinitrc

The **startx** shell script will first search for a file specified by the user's **XINITRC** environment variable. If this environment variable is not set (it is not set by default), then **startx** searches the user's **\$HOME** directory for a file called **.Xinit**, **.xinit**, **.Xinitrc**, **.xinitrc**, or **.xsession** respectively, to begin the X Client programs. If the file is not found in the user's **\$HOME** directory, the system-wide **/usr/lpp/X11/defaults/xinitrc** is used.

Creating a user .xinitrc file

If a user wishes to customize their own AIXwindows startup environment, they should copy the system-wide file into their **\$HOME** directory, rename it to make it a hidden file, and modify it. The file itself indicates where modifications should take place.

Example modifications

In the example on the visual the `xclock` command has been modified. The option `-update 1` indicates an update frequency of 1 second, which shows a second hand in the clock.

The `.xinitrc` shell script starts commands such as `xclock`, `aixterm`, and `mwm`. Note that the windows are started in the background. Only the last command, `mwm`, is started in the foreground.

Instructor Notes:

Purpose — Introduce the `.xinitrc` file.

Details — The graphic shows only a portion of the file. However, the portion shown is the part that a user would most likely tailor.

In the file, any line beginning with a # is a comment line.

In our example, the `xclock` is started first in the background. It is now time to begin discussion of how these windows can be tailored. In the system-wide file, the `xclock` uses a foreground color (`-fg`) of antique white and a background color (`-bg`) of grey. The hands of the clock (`-hd`) will be cadet blue, while the outline of the hands (`-hl`) are antique white. The flag `-geometry` indicates where on the screen the window is to reside. A geometry of `-0+0` indicates the window will reside in the upper right corner of the screen (Geometry will be covered in more detail shortly). As a suggestion, users could add the option of `-upgrade 1`. This will create a second hand that will be updated every second.

The next command to execute is `xsetroot`. This command defines the root window. As a suggestion, black could be used instead of grey. Some users find that a grey background flickers, and the black background seems easier on the eyes. Users will need to experiment with what works best on their terminal. We will discuss the `xsetroot` command in more detail later in the unit.

The `aixterm` command executes in the background next. The geometry of this window indicates that it will be 80 columns wide and 25 rows high. The geometry location of `+0-0` indicates the window will be located in the lower left corner of the screen. If a user wishes to have more than one `aixterm` upon AIXwindows startup, this is the place to add additional lines to define the other `aixterm` windows. If a user wishes to have a scientific calculator displayed, again this is the place to add such a line (for example, `xcalc &`).

The last line in the file to execute is the Motif window manager. Remind the students to watch AIXwindows come up. They will see the window displayed first, then Motif executes to add the frame to the window. On faster systems, the students will have to pay attention as this can happen very quickly!

Additional Information — While the AIXwindows programs tends to use the Motif window manager (`mwm`), there are actually a number of other window managers that exist, and can be used.

For example, if the `mwm` configuration file is not found, `startx` will attempt to look for another window manager to start. In the order of search, `startx` will look for:

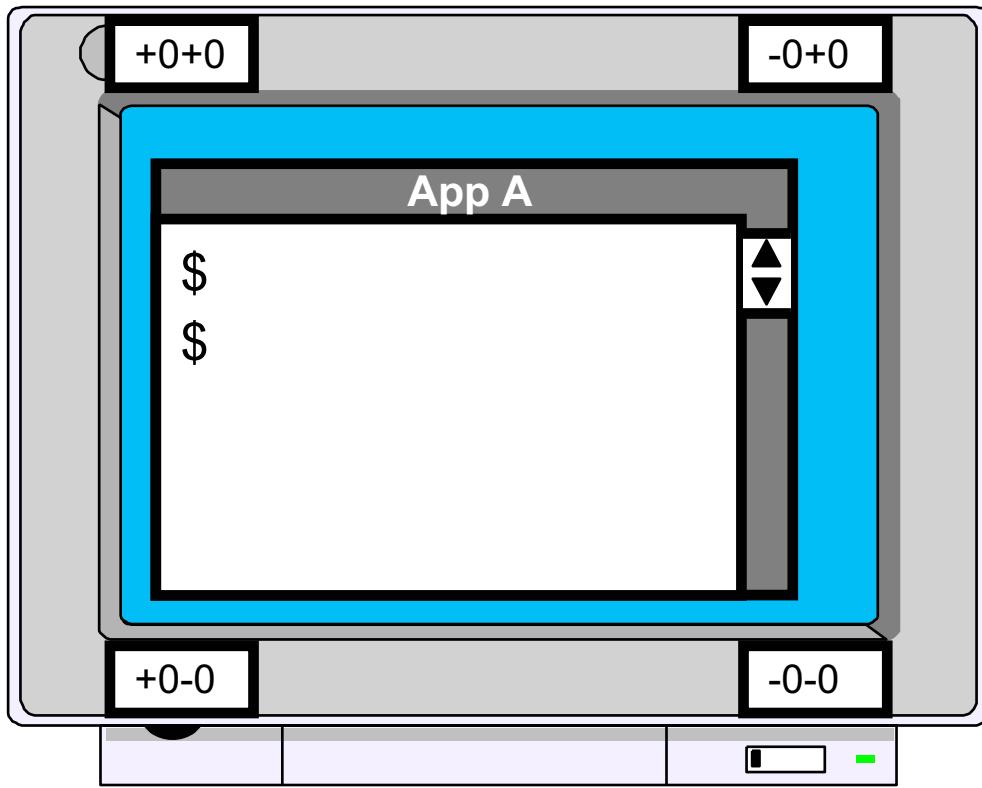
- `twm`: Tab window manager
- `awm`: Ardent window manager
- `uwm`: A window manager for X

It is possible that a student may use a window manager other than `mwm` or one of the window managers listed above. The window manager is what allows the user to manipulate the windows. Each window manager just works a bit differently.

The current version of the default **xinitrc** file is to run a script called **xstartterm** which, if CDE is installed, will start **dtterm** as the terminal emulator. If you prefer the **aixterm** emulator, you have two choices. You may create and customize your own **\$HOME/.xinitrc** or you can set the **XTERMINALEMULATOR** environment variable to **/usr/bin/aixterm**.

Transition Statement — Before we discuss the other files, let's look at some of the popular specifications that can be used to define a window: color, geometry, and fonts. These items are also used in the **.Xdefaults** file.

Geometry Specifications for Clients



© Copyright IBM Corporation 2008

Figure C-4. Geometry Specifications for Clients

AU1310.0

Notes:

Changing window size and location

One of the advantages of using the X Window system is that clients are not restricted to a particular size or location on the screen. Most X Clients accept a command line argument **-geometry WIDTHxHEIGHT +XOFF+YOFF** (where **WIDTH**, **HEIGHT**, **XOFF**, and **YOFF** are numbers).

Windows can be placed in the four corners of the screen by using the following specifications:

- +0+0 Upper left hand corner of the screen
- 0+0 Upper right hand corner of the screen
- +0-0 Lower left hand corner of the screen
- 0-0 Lower right hand corner of the screen

The **WIDTH** and **HEIGHT** specifications are usually measured in either pixels or characters depending on the application. A positive **XOFF** means an offset from the left

side of the screen. A negative value means an offset from the right side of the screen. A positive YOFF means an offset from the top of the screen. A negative value means an offset from the bottom of the screen.

These values are used in X resource statements and command line options. Specifying resources:

xclock.width:	200
xclock.height:	250
xclock.geometry:	-0+0

Examples

Examples of specifying the command line geometry option are:

```
aixterm -geometry 80x40+200+300  
xclock -geometry 200x250-0+0
```

In the above examples, the **aixterm** will consist of 80 rows by 40 columns of characters and be positioned 200 pixels from the left edge of the screen and 300 pixels from the top of the screen relative to the left corner of the client.

Similarly, the **xclock** will appear in the top right corner of the screen and will be 200 pixels in width and 250 pixels in height.

Instructor Notes:

Purpose — Describe some of the more common specifications used to define an AIX window.

Details — The size and position of a window on a display is referred to as the *window geometry*. The height and width of a window is usually expressed in *pixels* (*picture elements*). However, some applications may use units that make sense for that application. For example, the `aixterm` uses characters and rows to represent window size.

Like points on a graph, each window has its own coordinates. The x,y coordinates of each window's upper left corner, or origin is (0,0). The x coordinate increases as you move to the right within the window, and the y coordinate increases as you move down within the window.

The easiest way to determine what the location pixels (that is, x,y coordinates) should be is just to create the window and move it to where you want it to reside on the screen. When moving a window, a box is displayed which shows the window's x,y coordinates. Once you know the x,y coordinates, you can edit these values directly into one of the customization files.

Additional Information —

Transition Statement — Now that we understand geometry, let's look at what our color options are.

The Color Database

- The file **/usr/lib/X11/rgb.txt** contains a list of **valid colors**:

112	219	147	aquamarine
50	204	153	medium aquamarine
50	204	153	MediumAquamarine
0	0	0	black
0	0	255	blue
95	159	159	cadet blue
95	159	159	CadetBlue

- The **X Server** loads this **color database** by default
- To **view** and **select** valid colors use the command:

```
$ custom -e color
```

© Copyright IBM Corporation 2008

Figure C-5. The Color Database

AU1310.0

Notes:

The AIXwindows color database

The **rgb.txt** file associates RGB values with actual color names. These are the color names that you can use when customizing AIXwindows files or specifying command line options.

An RGB database is already built and the file **/usr/lib/X11/rgb.txt** lists the valid color names that can be specified as command line options or within customization files such as **.xinitrc** or **.Xdefaults**.

The numbers to the left of the color name indicate the degree of red, green, and blue in that color.

Instructor Notes:

Purpose — Explain the AIXwindows color database.

Details — There are many colors to choose from when customizing windows. The colors are listed in the database file `/usr/lib/X11/rgb.txt`. This file is loaded into the X Server when AIXwindows is started. It is possible to create additional color databases, but doing so is way beyond the scope of this class.

The file indicates three numbers, and then the name of the color. The three numbers indicate the mix of red, green, and blue respectively. You will note that many colors are listed more than once. If using a command to start a window, it will be necessary to use the name of the color without the spaces. For example, if you were starting an `aixterm` from the command line, it would be necessary to use a specification such as `CornflowerBlue` rather than `cornflower blue`.

A more interesting way to view the colors is with the `custom -e color` command. Have the students try this. They will see a graphics window displayed. They can scroll through the available colors. To see one of the colors displayed, just click the color. Colors can also be chosen using the red, green, and blue sliders on the screen. To do this, hold down the left mouse button and move the red, green and blue sliders. Once that is done, click **Match RGB to the Closest Color Name**.

We will discuss the custom application in more detail shortly and students will have a chance to play with this in the machine exercise.

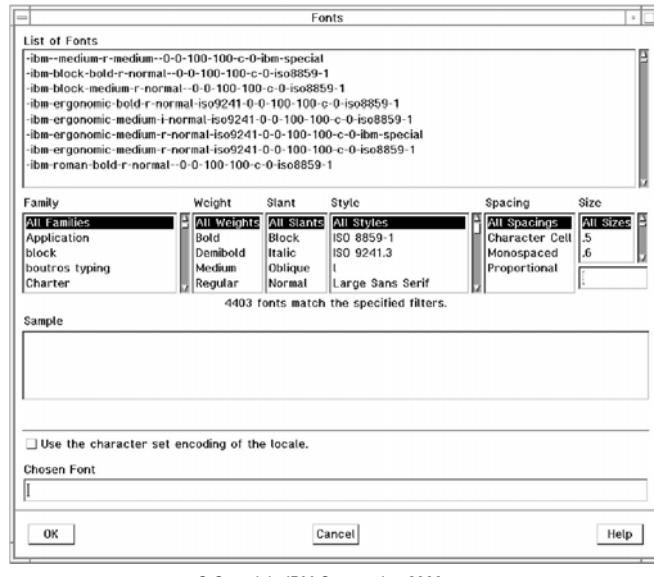
Additional Information — While modifying the color database is outside the scope of this class, modifying the `rgb.txt` file will not change anything. The system uses a BSD indexed database to hold the effective definitions. The purpose of examining the `rgb.txt` file is to know what color definitions are available to be referenced in one's customization of an application, provided one is not using the preferred custom tool (covered later in this unit).

Transition Statement — Now, let's look at the available fonts.

Fonts

- Fonts are stored in the directory **/usr/lib/X11/fonts**
- To **list all the fonts** available use the command:

```
$ custom -e font
```



© Copyright IBM Corporation 2008

Figure C-6. Fonts

AU1310.0

Notes:

Available fonts

Some font names are very simple, such as `rom10`. Other fonts names are made up of several hyphenated parts. The 13 hyphenated parts specify:

- | | |
|--|---|
| <ul style="list-style-type: none"> • foundry • font family • weight • slant • set width • additional style • pixels | <ul style="list-style-type: none"> • points • horizontal resolution • vertical resolution • spacing • average width • character set |
|--|---|

Instructor Notes:

Purpose — To show the many fonts that are available.

Details — Many fonts are delivered with AIXwindows. The easiest way to view the fonts is again by using the `custom -e font` command. More detail on the custom capabilities will be covered later. For now, however, you can have the students try this command.

Once the students have the font window displayed, have them scroll through the list of fonts at the top of the window. They can narrow the list by choosing the font family, weight, slant, spacing and size. Once the list is narrowed down, they can click an interesting font. If the font is loaded, it will be displayed in the box labeled `Sample`.

Students will have plenty of opportunity to play with the fonts in the machine exercise.

Additional Information —

Transition Statement — Now that we have covered some of the common definitions we use for windows, let's move ahead and resume the discussion of the startup files. The next file we will discuss is the `.Xdefaults` file.

.Xdefaults

Use **.Xdefaults** to **customize** your AIXwindows.

```
$ vi $HOME/.Xdefaults
```

```
Aixterm*background:      grey
Aixterm*foreground:      navy
Aixterm*font:            rom10
xclock*update:           1
Mwm*keyboardFocusPolicy: explicit
```

© Copyright IBM Corporation 2008

Figure C-7. .Xdefaults

AU1310.0

Notes:

The .Xdefaults file

Most of the customization of AIXwindows is done through the use of *resources*. A resource is a way to specify the default behavior for a type of window or for the Motif window manager. For example, the background color for any **aixterm** window is considered a resource. The preferred focus policy or the decision to always use a scrollbar with an **aixterm** are also resources.

Setting AIXwindows resources

Most resources are set in a user's **.Xdefaults** file. This file can be created using a text editor or by using the AIXwindows *custom* application. The *custom* application will be discussed in more detail shortly. The Motif window manager will read this file during its startup process.

Each resource specified will look something like this:

object*attribute: value

Object is the name of the program, such as **aixterm**. Attribute is the resource associated with the program, such as geometry, font, or background. Value is the value assigned to the attribute, such as specifying that the background color will be grey. For example:

Aixterm*background: grey
Mwm*keyboardFocusPolicy: pointer

When adding entries by hand, be sure there are no trailing blanks after any of the lines. To verify this, use the **cat -vte** command to display the file. In the output, the \$ indicates a carriage return.

To view a definition for each of the attributes available for an **aixterm**, execute the command: **aixterm -keywords | pg**.

Instructor Notes:

Purpose — Introduce the **.Xdefaults** file and the use of resources.

Details — If users wish to customize their AIXwindows environment, the **.Xdefaults** file is the place to do it. The **.Xdefaults** file is often referred to as the X resource file. The file is located in the user's `$HOME` directory and can be created and later edited using a text editor or via the AIXwindows *custom* application. We will look at the *custom* application shortly. Students will have a chance to use it during the machine exercise.

A thorough discussion of the **.Xdefaults** file could take days. There are many options available in the way each resource can be coded. We want to just cover the basics here.

This file can include specifications for `aixterm` windows, `xclock`, `xcalc`, as well as `mwm` features such as focus policy and the behavior of the root and window menus. Any windows that are created will use the specifications indicated in this file.

Within the file, an asterisk (*) acts as a wildcard name holder. On the visual, the * indicates that every `aixterm` that is created will have a background of grey, foreground of navy and font of `rom10`. If a resource is specified twice in error, for example, two lines indicating the background color of an `aixterm`, the second line read will be the one used.

Try not to get into a discussion of the difference between using an * and a . to separate the object and the attribute. Surprisingly, the explanation is complex. However, suffice it to say that if a resource is specified twice, first using the . and the second entry using the *, the entry with the . will prevail.

If a user types the `aixterm` command on the command line, they can override the specifications listed in the **.Xdefaults** file.

Any line in the file beginning with an ! is a comment.

If a user enters resources into the file by hand, tell them not to forget the : after the attribute. This is a very common error to make when editing this file by hand. If editing **.Xdefaults** by hand it is recommended to make a copy and then comment any lines that will be changed. Thus, if the changed line does not work, you can uncomment the previous version of the command.

AIX V4.1 and V4.2 provide a template that can be used to create a customized **.Xdefaults** file. This sample file is called `/usr/lpp/X11/defaults/Xdefaults.tmpl`. AIX 5L V5.1 and onward do not include a template for this file. This is not a problem in that most users will probably use the *custom* application to build their **.Xdefaults** file.

Additional Information — Actually, the `startx` program will look for resource definition in the following files: **.Xdefaults**, **xdefaults**, **.Xresources**, and **.xresources** respectively. If one file is not found, the search will continue for the next file, and so forth. In the AIXwindows world, the **.Xdefaults** file is the most common file to use. However, students may use one of the other customization files in their own environments instead.

Transition Statement — The last file we need to discuss is **.mwmrc**. This file is also executed by the Motif window manager.

.mwmrc

```
$ cp /usr/lpp/X11/defaults/Motif1.2/system.mwmrc $HOME/.mwmrc
$ vi $HOME/.mwmrc
```

```
Menu DefaultRootMenu
{
    " Root Menu "          f.title
    no-label                f.separator
    " New Window "         f.exec "aixterm"
    " My Window "          f.exec "aixterm -bg white -fg navy"
    " Clients "             f.menu "clients"
    " Custom "              f.exec "custom"
    no-label                f.separator
    " Refresh "              f.refresh
    " Pack Icons "          f.pack_icons
    no-label                f.separator
    " Restart ... "         f.restart
    " Quit ... "             f.quit_mwm
    no-label                f.separator
    " End Session "        f.menu "end_session"
}
~
~
:wq
```

Do not directly edit the system-wide system.mwmrc!

© Copyright IBM Corporation 2008

Figure C-8. .mwmrc

AU1310.0

Notes:

Further customization

Most of the features that you want to customize can be set with resources in a user's **.Xdefaults** file. However, root menu options, window menu options and the behavior of the mouse can be customized in a file called **.mwmrc**.

The system .mwmrc file

In AIX V4.1 and V4.2, the system-wide copy of this file is located in **/usr/lib/lpp/X11/system.mwmrc**. In AIX V4.3 and all subsequent releases, the file can be found in **/usr/lpp/X11/defaults/Motif1.2/system.mwmrc**. Do not modify the system-wide files.

Custom version of the .mwmrc file

If you wish to customize any of this information for your own environment, copy the system-wide file into your \$HOME directory. Rename the file **.mwmrc**. Your own **.mwmrc** file will override the system-wide version for your AIXwindows environment.

In the example a line has been added to **\$HOME/.mwmrc**:

```
" My Window "    f.exec "aixterm -bg white -fg navy"
```

This line adds a new item **My Window** to the root menu. When this item is selected, a customized **aixterm** will be started.

Instructor Notes:

Purpose — Explain the use of the **.mwmrc** file.

Details — In most cases, a user will not need to customize this file. However, there may be instances when they do want to make a change to the root menu, the window menu or the behavior of the mouse buttons.

If changes are to be made, the system-wide file must be copied to the user's **\$HOME** directory and renamed as **.mwmrc**.

As an example of a change to this file: Suppose the user wanted to have a root menu option for a customized **aixterm** window. Within the file, add a line to the section called **Menu DefaultRootMenu**. The line would look something like this:

```
" My Window "    f.exec "aixterm -bg white -fg navy "
```

Once AIXwindows is restarted, the new menu item will be in effect.

In case there is a question, the Motif window manager menus cannot be customized using the AIXwindows *custom* tool.

Additional Information — If another window manager (other than the Motif window manager) is being used, another customization file will be used instead. For example, if the Ardent window manager (**awm**) is being used, it will use the **.awmrc** file rather than **.mwmrc**.

Transition Statement — Let's have an exercise.

Exercise: Customizing AIXwindows (1)



© Copyright IBM Corporation 2008

Figure C-9. Exercise: Customizing AIXwindows (1)

AU1310.0

Notes:

After completing the exercise, you will be able to:

- Customize the **.xinitrc** file
- Customize the **.Xdefaults** file

Instructor Notes:

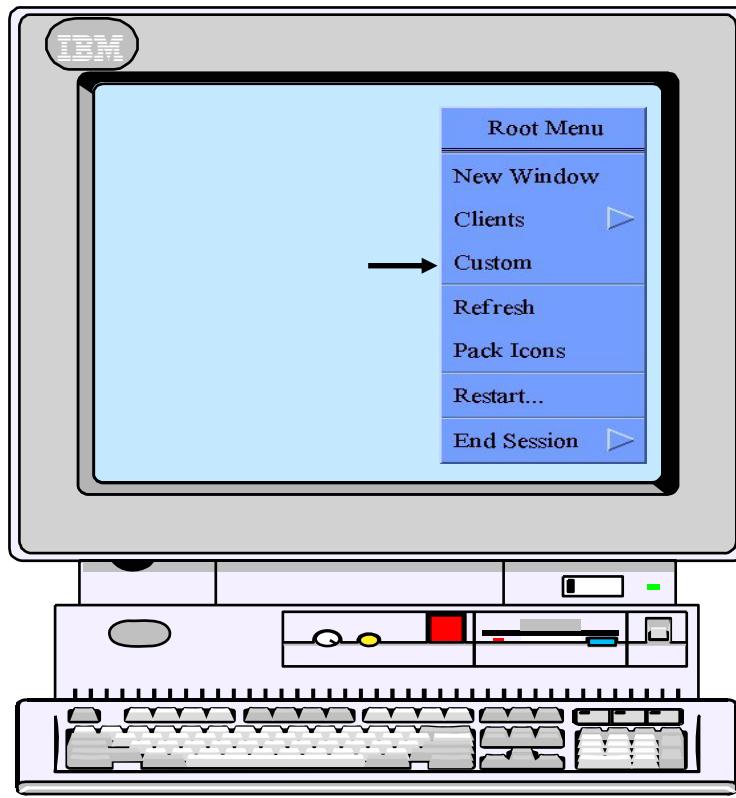
Purpose — Pointer to next exercise.

Details — Provide what students will learn.

Additional Information —

Transition Statement — We now introduce working with the `custom` command.

AIXwindows Custom Application



© Copyright IBM Corporation 2008

Figure C-10. AIXwindows Custom Application

AU1310.0

Notes:

Making AIXwindows changes with the custom application

To begin the AIXwindows *custom* application, in the AIXwindows environment, click the right mouse button in the root window. The root menu will appear (remember, this is the menu that can be customized using the `.mwmrc` file). While continuing to hold down the right mouse button, point to the *Custom* option and release the mouse button. The Customizing Tool window will appear.

Another possibility to start the *custom* tool, is to execute the `custom` command in a window.

Instructor Notes:

Purpose — Show the students how to access the AIXwindows *custom* tool.

Details — Explain to the students how to access the root menu and choose the custom option.

You may wish to remind the students that this root menu can be customized by editing the **\$HOME/.mwmrc** file.

Additional Information —

Transition Statement — Once the custom option is chosen, the Customizing Tool window will be displayed. We will look at this menu next.

The Custom Window

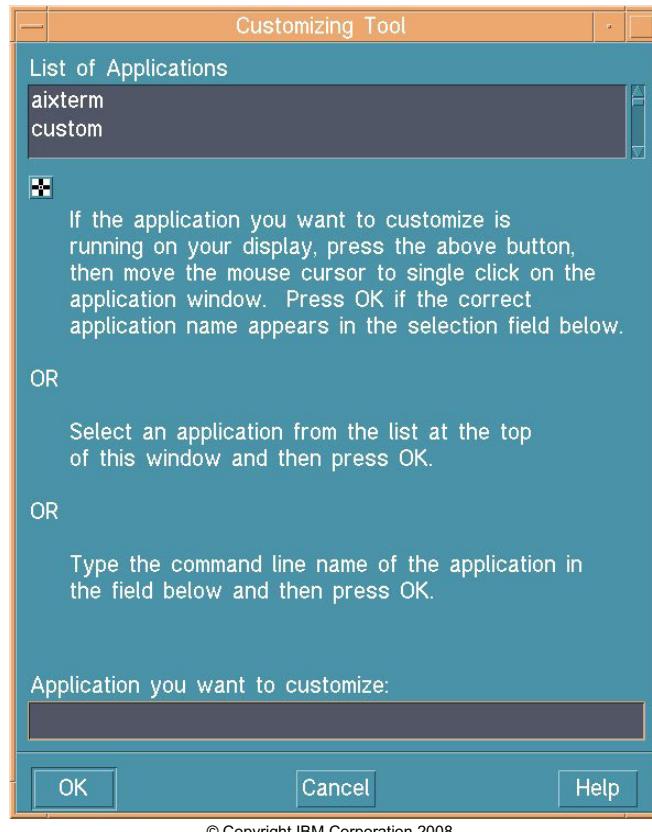


Figure C-11. The Custom Window

AU1310.0

Notes:

The Customizing Tool window

The Customizing Tool window allows a user to customize their own AIXwindows environment. Most items chosen for customization will be placed in the user's **\$HOME/.Xdefaults** file.

Note the list of applications at the top. A scrollbar is available to view the entire list of applications. Use the left mouse button to click the application that needs to be customized. Then, click **OK**.

Each application has its own set of resources that can be customized. This list can be found in the **/usr/lib/X11/app-custom** directory. In the directory is a filename for each of the applications listed on Customizing Tool window. This file describes what can be modified and the possible range of values. If a filename does not exist, the application will use the resources listed in a file called **DEFAULT**.

Instructor Notes:

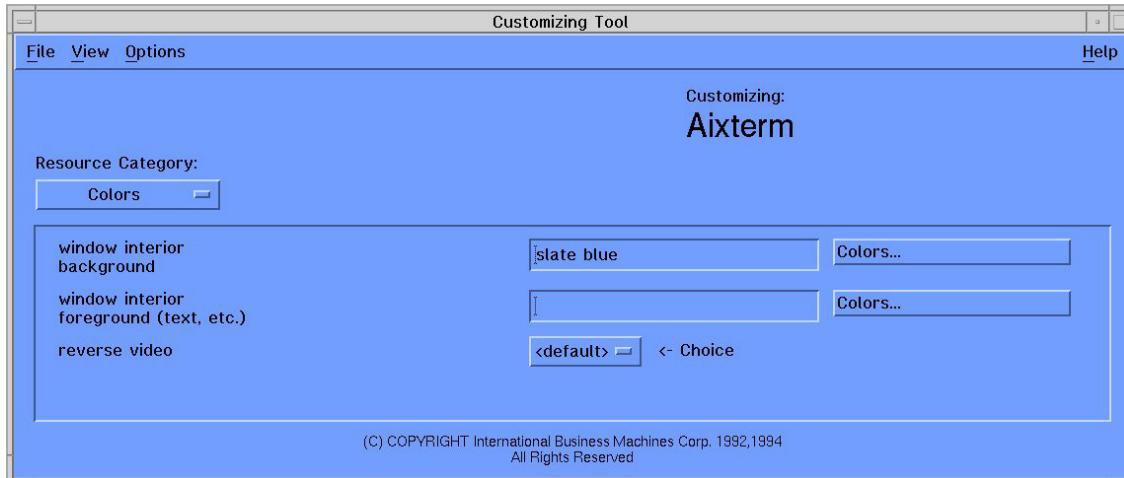
Purpose — Explain the AIXwindows *custom* window.

Details — The interface is straightforward to use. Use the left mouse button to click the application you wish to customize and then click OK.

Additional Information —

Transition Statement — Let's assume we wish to modify the default behavior of our `aixterm` windows. Click `aixterm` and then `OK`. The following window will be displayed.

Customizing an `aixterm`



© Copyright IBM Corporation 2008

Figure C-12. Customizing an `aixterm`

AU1310.0

Notes:

Introduction

Above is the window a user would see if customizing an `aixterm` window. The windows to customize other applications such as the `xclock` or `mwm` would look similar.

Resource Categories

Note the Resource Category selection area. For an `aixterm`, the possible resources that can be customized are colors, fonts, size and location, icon, graphics (includes window title and cursor characteristics), scrollbar, and behaviors. Other applications will include different resource categories.

Customizing colors

The example above shows the window displayed if customizing color selections for an **aixterm**. Note that three options are available. Clicking colors will display the color browser.

The above window can also be accessed from the command line by typing:

```
$ custom aixterm
```

Instructor Notes:

Purpose — Introduce the Customizing Aixterm window.

Details — This is the initial window for customizing **aixterm** windows. Use the left mouse button to click the color Resource Category. Additional customizable resource categories will be displayed, such as fonts, size and location, icon, graphics, scrollbar, and behaviors.

As we continue our example, let's assume we wish to change the background color of our **aixterm** windows. To the right of the input box for that item is a box, **Colors...**. Click this box with the left mouse button. We will look at the color browser next.

Additional Information —

Transition Statement — Once we click **Colors...**, the color browser will be displayed. This is what we will look at next.

The AIXwindows Color Browser

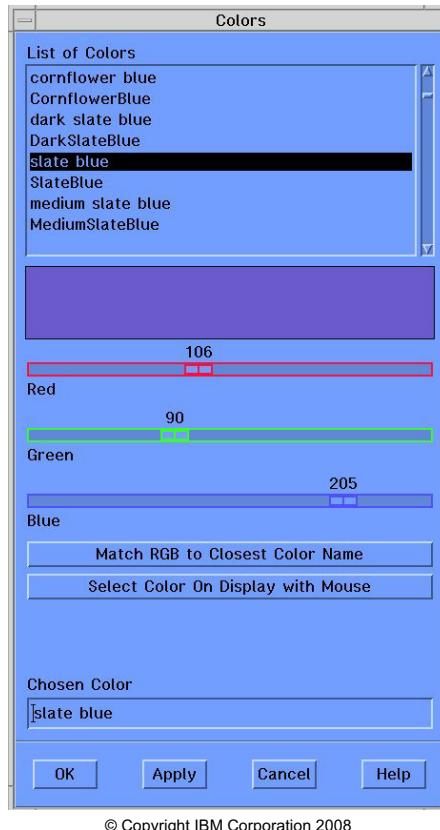


Figure C-13. The AIXwindows Color Browser

AU1310.0

Notes:

Color browser

Above is the AIXwindows color browser. In the top window is a list of colors. A scrollbar is available to view all the possible colors. The colors listed are from the color database that was discussed earlier, file `/usr/lib/X11/rgb.txt`.

Choosing colors

Colors can be chosen a couple of ways. One way is to scroll the list of colors. When a color looks interesting, use the left mouse button to click the name of the color. The actual color will be displayed. Note the sliders for red, green, and blue. These will change as well to indicate the mixture of these primary colors. Click `OK` to indicate that you have chosen your color for the specific resource.

Another way to choose a color is to use the left mouse button to actually move the sliders for red, green, and blue. Then, click Match RGB to Closest Color Name. The

name of the color will be displayed in the Chosen Color box and the color will be displayed in the window. Again, if you like the color, use the left mouse button to click OK and this is the color that will be used for the resource.

Other browsers

Similar browsers also exist for fonts, the cursor and pictures (bitmaps).

These browsers can also be accessed directly from the command line with the **custom -e** command. For example, the command **custom -e cursor** will display a browser that allows you to select the look of the cursor.

Instructor Notes:

Purpose — Discuss how the AIXwindows browsers work.

Details — The browsers are easy to work with and the students will have the opportunity to use them during the machine exercise.

Additional Information —

Transition Statement — Now, assume you have made some changes to one of the AIXwindows resources (such as the background color of an `aixterm`) Now, the changes will need to be saved. We will discuss how to do that next.

Saving the Customized Changes



© Copyright IBM Corporation 2008

Figure C-14. Saving the Customized Changes

AU1310.0

Notes:

How to save changes

To save any changes that are made, back on the original customizing menu for the window, use the left mouse button and click **File** in the upper left corner of the window.

The **File** menu will appear. Choose the option **Save As...**. Generally, you can just click **OK** on the **Save As...** screen. This will immediately save any resource updates to your **\$HOME/.Xdefaults** file. Then choose **File** again and then **Exit** to close the customizing tool.

At this point, if you created a new window, the changed resource value would be used. Later, if starting AIXwindows using the **startx** command, the new values will be used for any windows.

It is important to note that now the Customizing Tool is updating the **.Xdefaults** file for you. You can still manually edit this file if you wish.

Instructor Notes:

Purpose — Describe how to save any resource changes.

Details — Cover the information in the student notes describing how to save changes that are made.

You may wish to point out the file **Xdefaults.bak**. This file is a copy of the previous version of the **Xdefaults** file in case you need to fall back to it. CDE will automatically save this file each time changes are made to **Xdefaults**.

Additional Information —

Transition Statement — One last item may need to be changed and that is the root window. This will be discussed next.

The **xsetroot** Command

Customize the **root window** using the **xsetroot** command.

```
$ xsetroot -solid black
```

```
$ xsetroot -cursor_name gumby
```

```
$ xsetroot -bitmap /usr/include/X11/bitmaps/xsnow
```

Permanently customize the root window in **.xinitrc**.

© Copyright IBM Corporation 2008

Figure C-15. The **xsetroot** Command

AU1310.0

Notes:

Introduction

The **xsetroot** command is used to tailor the appearance of the root window on a workstation running AIXwindows.

Changing the default root window

The default root window is a speckled grey, but you may want something a bit more snazzy. It is typical to first experiment with the **xsetroot** command until you come up with a look you like. Then, put the finalized **xsetroot** command into your **.xinitrc** file. You can change characteristics such as the color, design (bitmap), and pointer cursor.

When experimenting with the **xsetroot** command, use the **xsetroot -def** command to reset the root window back to its default values.

When the **xsetroot** command is executed from the command line, changes take place immediately.

xsetroot command options

Below are some of the available options with **xsetroot**:

-bg <Color>	Specifies the color for the window background.
-bitmap <FileName>	Use the specified file name as the bitmap image.
-cursor <CursorFile> <MaskFile>	Specifies the cursor and mask files so that the pointer can be changed whenever the pointer is outside any window.
-cursor <CursorName>	Sets the pointer cursor to one of the standard cursors from the cursor font.
-def	Reset attributes to the default values.
-display <Name:Number>	Identifies the host server name and the X Server display number where the command is to run. If this is not specified, the client program gets the host name and display number from the <i>DISPLAY</i> environment variable.
-fg <Color>	Specifies the color for the window foreground.
-grey	Makes the entire background grey in color.
-help	List the available option flags.
-mod <x,y>	Produces a plaid like grid pattern. The X and Y parameters are integers ranging from 1 to 16.
-rv	Reverses the foreground and background colors.
-solid <Color>	Sets the background of the root window to the specified color.

Note that bitmaps can be viewed using the **cvt2x11 -e picture** command.

Instructor Notes:

Purpose — Discuss the `xsetroot` command.

Details — The graphic shows three examples of the `xsetroot` command:

```
xsetroot -solid black
```

The default background of the root window is a speckled black and white herringbone pattern, which looks grey. Some users find this hard on the eyes, so using a solid color may help.

```
xsetroot -cursor_name gumby
```

The `-cursor_name` option changes the pointer cursor. The available patterns that can be used can be viewed using the `custom -e cursor` command. The `xfd -fn cursor` command can also be used to display available patterns, but it does not display the cursor name. The pointer cursor is quite small, so it may be difficult to actually see that it is gumby or a skull and crossbones, and so forth.

```
xsetroot -bitmap /usr/include/X11/bitmaps/xsnow
```

Bitmaps are black and white images whose file names end in `.xbm` or `.bm`. Available bitmaps can be found in the `/usr/include/X11/bitmaps` directory. The above command changes the grey root window to display snowflakes. Use the `-bg` and `-fg` options to change the colors used by this image. It is also possible to use the `bitmap` command to create your own bitmap. We will not be doing this in this class, but students are welcome to try it on their own (that is `bitmap cat`).

Bitmaps can also be viewed using the `custom -e picture` command.

Additional Information —

Transition Statement — Before we move to the second lab in this unit, let's review a few checkpoint questions.

Checkpoint

1. Match the AIXwindows startup file with its function:

a) .xinitrc	_____ Sets default characteristics for AIXwindows resources
b) .Xdefaults	_____ Starts the Motif window manager
c) .mwmrc	_____ Defines the function of the root menu and the window menu
2. Name two ways the **.Xdefaults** file can be customized.
3. True or false? The AIXwindows custom tool saves all customization choices in the **.xinitrc** file.
4. What command is used to change the appearance of the root window?
5. Where would the **xsetroot** command be placed to make a permanent change to the root window?

© Copyright IBM Corporation 2008

Figure C-16. Checkpoint

AU1310.0

Notes:

Instructor Notes:

Purpose — To test the students' knowledge on this unit's material.

Details —

Checkpoint Solutions

1. Match the AIXwindows startup file with its function:

a) .xinitrc	<u>b</u> Sets default characteristics for AIXwindows resources
a) .Xdefaults	<u>a</u> Starts the Motif window manager
b) .mwmrc	<u>c</u> Defines the function of the root menu and the window menu
2. Name two ways the **.Xdefaults** file can be customized. **Edit manually, use the AIXwindows custom tool.**
3. True or false? The AIXwindows custom tool saves all customization choices in the **.xinitrc** file. **FALSE, most changes are stored in the .Xdefaults file.**
4. What command is used to change the appearance of the root window? **The xsetroot command.**
5. Where would the **xsetroot** command be placed to make a permanent change to the root window? **In .xinitrc.**

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — Let's do the second exercise that deals with customizing AIXwindows.

Exercise: Customizing AIXwindows (2)



© Copyright IBM Corporation 2008

Figure C-17. Exercise: Customizing AIXwindows (2)

AU1310.0

Notes:

After completing the exercise, you will be able to:

- Use the *custom* tool to tailor the AIXwindows environment
- Use the **xsetroot** command to customize the *root window*

Instructor Notes:

Purpose — Introduce the second lab in this unit.

Details —

Additional Information —

Transition Statement — Let's wrap this unit with a quick summary.

Unit Summary

- The **.xinitrc** file controls which windows to start during AIXwindows startup. This file starts mwm last.
- The **.Xdefault** file customizes various AIXwindows resources used by a user.
- The **.mwmrc** file customizes the root menu, the window menu, and the behavior of the mouse.
- Use the AIXwindows “**custom**” tool to customize the windows environment.
- The **xsetroot** command will customize the root window.

© Copyright IBM Corporation 2008

Figure C-18. Unit Summary

AU1310.0

Notes:

Instructor Notes:

Purpose — To summarize this unit's key points.

Details —

Additional Information —

Transition Statement — Let's move on to the next unit.

Appendix D. CDE User Customization

Estimated Time

01:15

What This Unit Is About

This unit provides the details needed for users to customize their CDE desktop working environment.

What You Should Be Able to Do

After completing this unit, you should be able to:

- Use the Style Manager to interactively customize the desktop environment
- Customize the Front Panel

How You Will Check Your Progress

Accountability:

- Student activity
- Checkpoint questions
- Exercise

References

SC23-2793 *CDE User's Guide*

SC23-2795 *CDE Advanced User's and System Administration Guide*

Unit Objectives

After completing this unit, you should be able to:

- Use the [Style Manager](#) to interactively customize the desktop environment
- Customize the [Front Panel](#)

© Copyright IBM Corporation 2008

Figure D-1. Unit Objectives

AU1310.0

Notes:

Instructor Notes:

Purpose — To introduce the unit objectives.

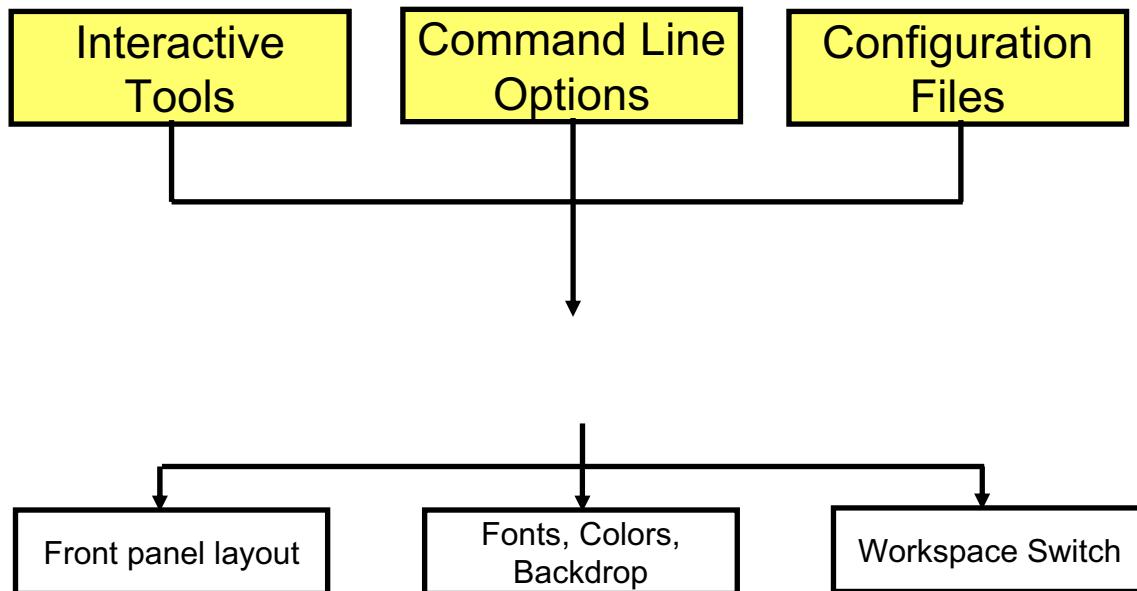
Details —

Additional Information —

Transition Statement — Let's look at the features that can be customized in CDE.

Customizing CDE

Most features of CDE are **customizable**



© Copyright IBM Corporation 2008

Figure D-2. Customizing CDE

AU1310.0

Notes:

What CDE features can be customized?

Most features of CDE are customizable. The front panel can be customized for each specific user. Users can also customize things like colors, fonts, and backdrops via the Style Manager. The Workspace Switch can also be changed and names can be given to each of the workspaces. Icons can also be updated or created.

There are several methods of customization. New users will find that the CDE interactive tools are the easiest way to customize CDE. More advanced users may wish to use command line options or directly update CDE configuration files.

Instructor Notes:

Purpose — Explain what can be customized in CDE and how the customization can be done.

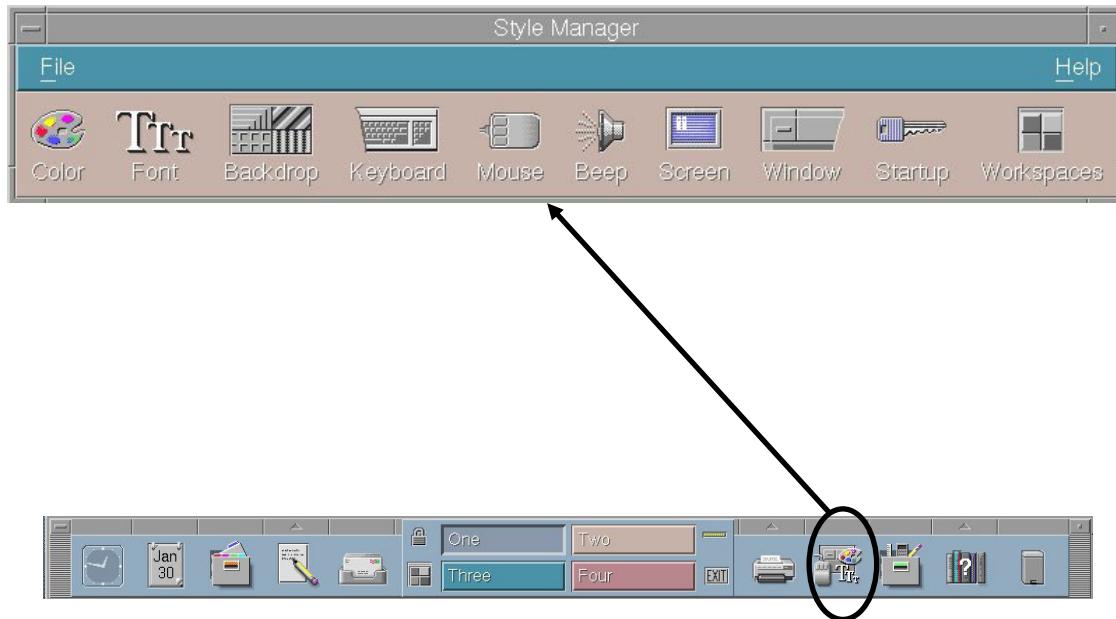
Details — Most of the basic CDE customization can be done using the CDE interactive tools. The Style Manager can be used to perform a number of the customization tasks. Some customization, the more advanced tasks, can only be performed by entering command line options or updating configuration files.

In this unit as well as in the machine exercise, we will focus on using the CDE interactive tools for customization. However, we will spend a small amount of time discussing the various CDE configuration files that can be tailored.

Additional Information —

Transition Statement — We will first look at the Style Manager.

Style Manager Overview



© Copyright IBM Corporation 2008

Figure D-3. Style Manager Overview

AU1310.0

Notes:

Style Manager customization

Many aspects of your session can be interactively changed with the CDE Style Manager. The Style Manager allows you to customize:

Color:	Workspace colors and palette
Font:	Application font sizes
Backdrop:	Workspace backdrop patterns
Keyboard:	Volume and character repeat
Mouse:	Settings, double-click settings, acceleration, and threshold
Beep:	Volume, tone, and duration
Screen:	Screen saver and screen lock
Window:	Focus policies and icon placement
Startup:	How your session begins and ends
Workspaces:	Show workspace buttons (default)

Instructor Notes:

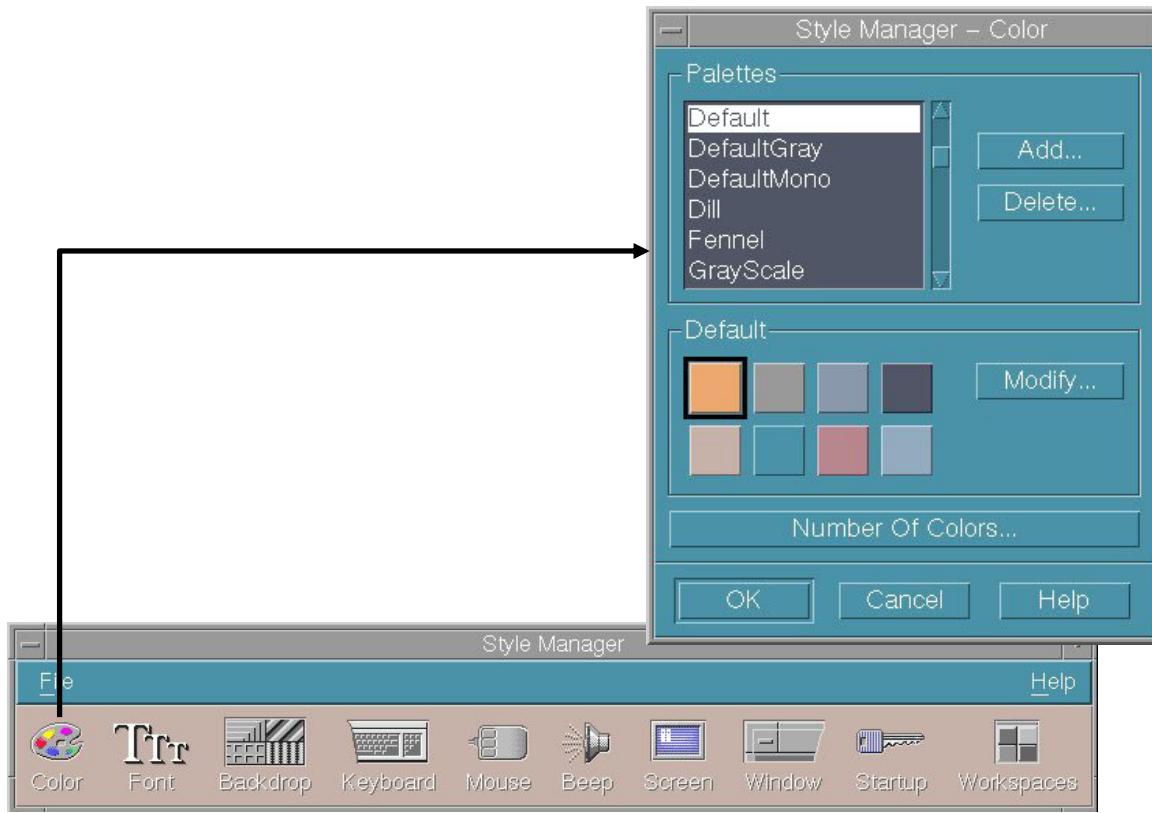
Purpose — Introduce the capabilities of the Style Manager.

Details — The Style Manager provides one of the easiest ways to customize CDE. We will look at each of the Style Manager controls shown on the visual.

Additional Information —

Transition Statement — We will first look at using the Style Manager to customize colors.

Style Manager - Colors



© Copyright IBM Corporation 2008

Figure D-4. Style Manager - Colors

AU1310.0

Notes:

CDE color palette

Workspace colors are set through a color palette. The number of color buttons in the palette is determined by your display type and the Number of Colors selection. Depending on the display, there may be two, four, or eight color buttons in the color dialog box. The color palette is used for screen characteristics such as the active and inactive window borders, text and list areas, main window background, front panel background, and so forth.

Number of Colors to Use also determines the number of colors in the color palette. The default is More Colors for Applications, which keeps the number of colors used on a high-color display to a minimum, thus saving the colors for applications.

Using the Style Manager, palettes can be created, modified, or deleted.

Instructor Notes:

Purpose — Discuss how the CDE color palette can be changed.

Details — The color palette is a family of colors that will be used for the CDE display. Using one of the existing palettes at least guarantees that the colors used for CDE will blend and be pleasing to the eye. Currently, students are using the default palette. The number of colors in the palette will vary depending on the type of display used and the value chosen for Number of Colors to Use. The color palette will consist of two, four, or eight colors.

In terms of Number of Colors to Use, the default is More Colors for Applications which keeps the number of colors used on a high-color display to a minimum. On the Number of Colors to Use window, the highest choice in the list provides the most colors to be used by CDE; the lowest choice (Black and White), provides the least number of colors to be used by CDE.

Now, why would you want to reduce the number of colors to be used by CDE? Suppose you are using CDE, but also running a color-rich application such as computer aided design (CAD). In this case, choose More Colors for Applications or Most Colors for Applications to decrease the number of colors that the desktop uses. The remaining colors will be used for the CAD program.

Additional Information —

Transition Statement — We'll next look at the Style Manager Font selection.

Style Manager - Fonts

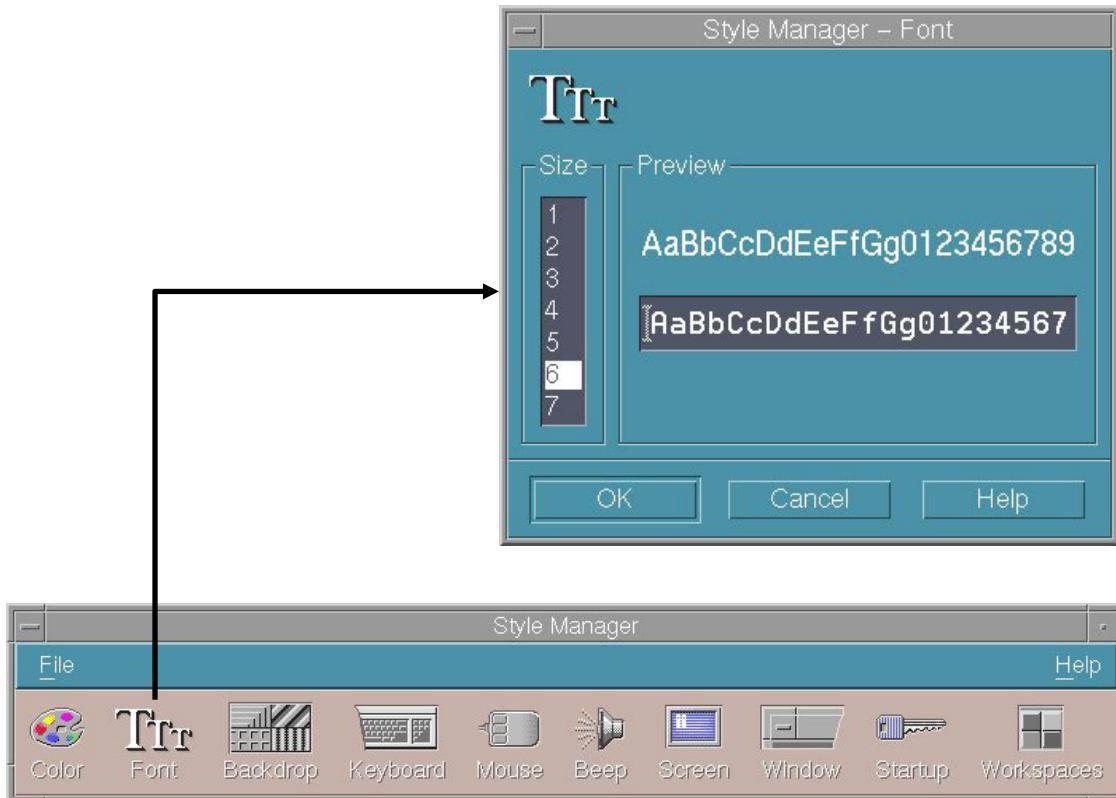


Figure D-5. Style Manager - Fonts

AU1310.0

Notes:

Choosing new fonts

The Style Manager font dialog allows you to select the font size used on window labels and menus.

Window labels and text will show the new font size the next time some applications are started. For other applications, such as the File Manager or the Application Manager, it will be necessary to exit CDE and then log back in to see the new fonts.

Instructor Notes:

Purpose — Introduce the Style Manager font control.

Details — To use the `Font` controls, merely click in the `Size` control box. The new font size will be shown in the `Preview` box. Once you find the font you like, click `OK`.

The new font size is used as applications are started. Existing windows will not reflect the change. To see the font used throughout CDE, it is best to exit CDE and then log back in.

Additional Information —

Transition Statement — The Style Manager can also be used to choose a backdrop for each workspace.

Style Manager - Backdrops

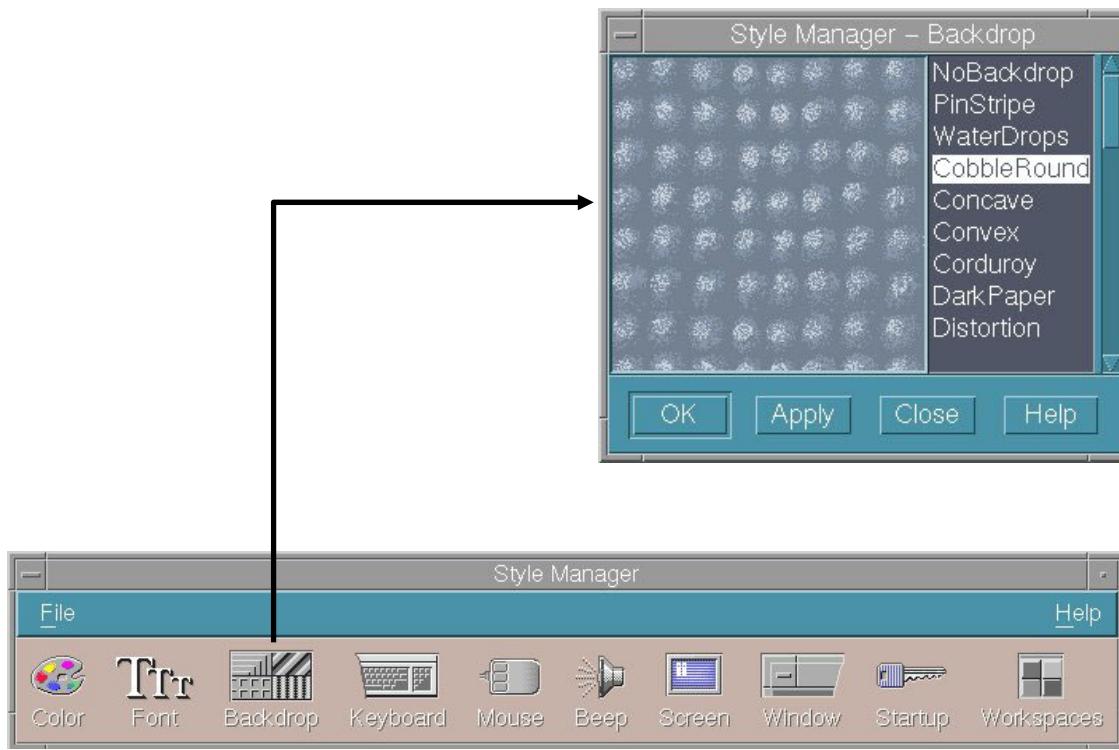


Figure D-6. Style Manager - Backdrops

AU1310.0

Notes:

Changing the workspace backdrop

Each workspace may have a characteristic pattern for the backdrop. A unique backdrop for each workspace adds variety and helps you to quickly identify the workspace you are in.

Instructor Notes:

Purpose — Show how the Style Manager can be used to choose a workspace backdrop.

Details — Within a workspace, choose a unique backdrop that will be used. To view the backdrop, merely click the backdrop name and the backdrop will be displayed. Once you find a backdrop that you like, click either **Apply** or **OK**.

If you have root authority, it is also possible to create your own backdrops. To do this, create a pixmap using the CDE Icon Editor and copy it to the directory, **/usr/dt/backdrops**.

Additional Information —

Transition Statement — Let's next look at how the Style Manager can customize the behavior of the keyboard, mouse, and beep.

Style Manager - Keyboard, Mouse, and Beep

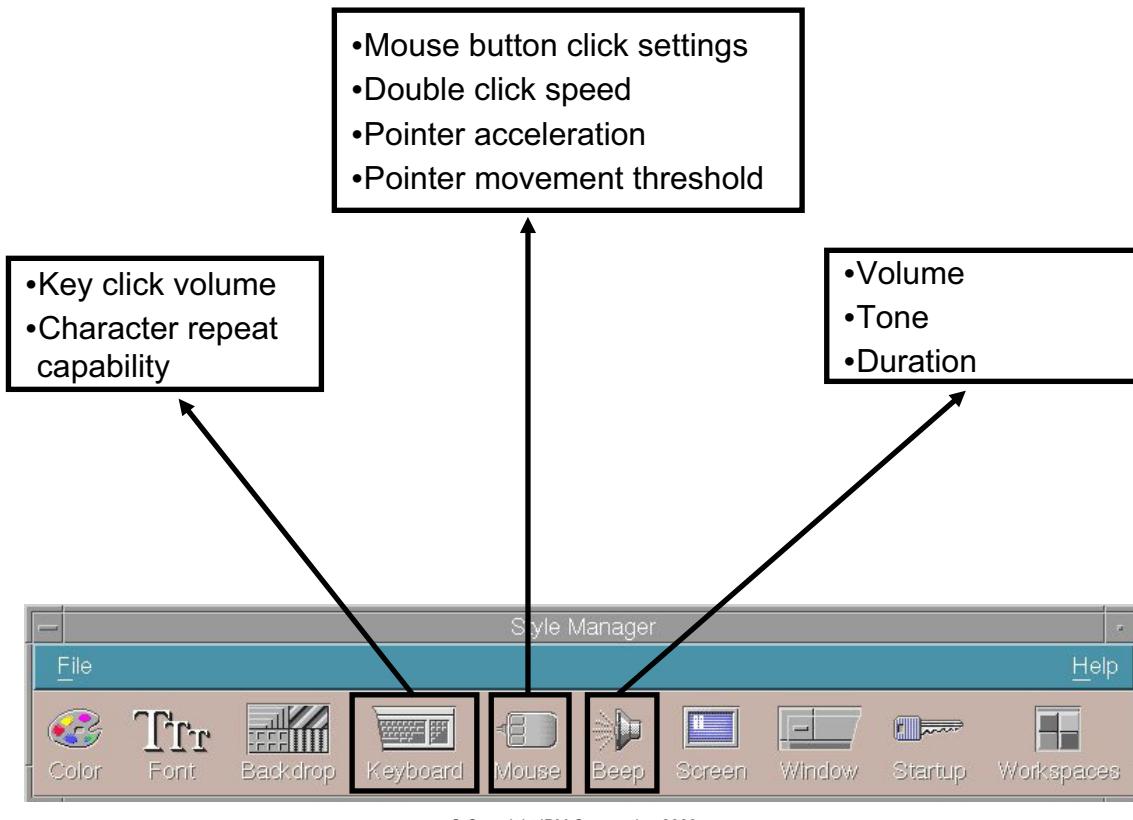


Figure D-7. Style Manager - Keyboard, Mouse, and Beep

AU1310.0

Notes:

Changing the behavior of the keyboard, mouse, and beep

The Style Manager allows the keyboard click volume and character repeat to be changed.

The mouse can be changed for right-or left-handed users (right is the default). The behavior of the middle mouse button can also be altered. Mouse acceleration refers to how fast the mouse pointer moves across the display. Pointer movement threshold refers to the distance in pixels the pointer moves at a slow speed before moving at the accelerated rate.

The beep volume can also be altered, where the range is 0 to 100%. 50% is the default and 0 means no volume. The tone is the frequency or pitch of the system beep, from 82 to 9000 Hertz (the default is 400). The duration of the system beep can be from .1 (the default) to 2.5 seconds.

Instructor Notes:

Purpose — It is also possible to change the behavior of the keyboard, mouse, and beep.

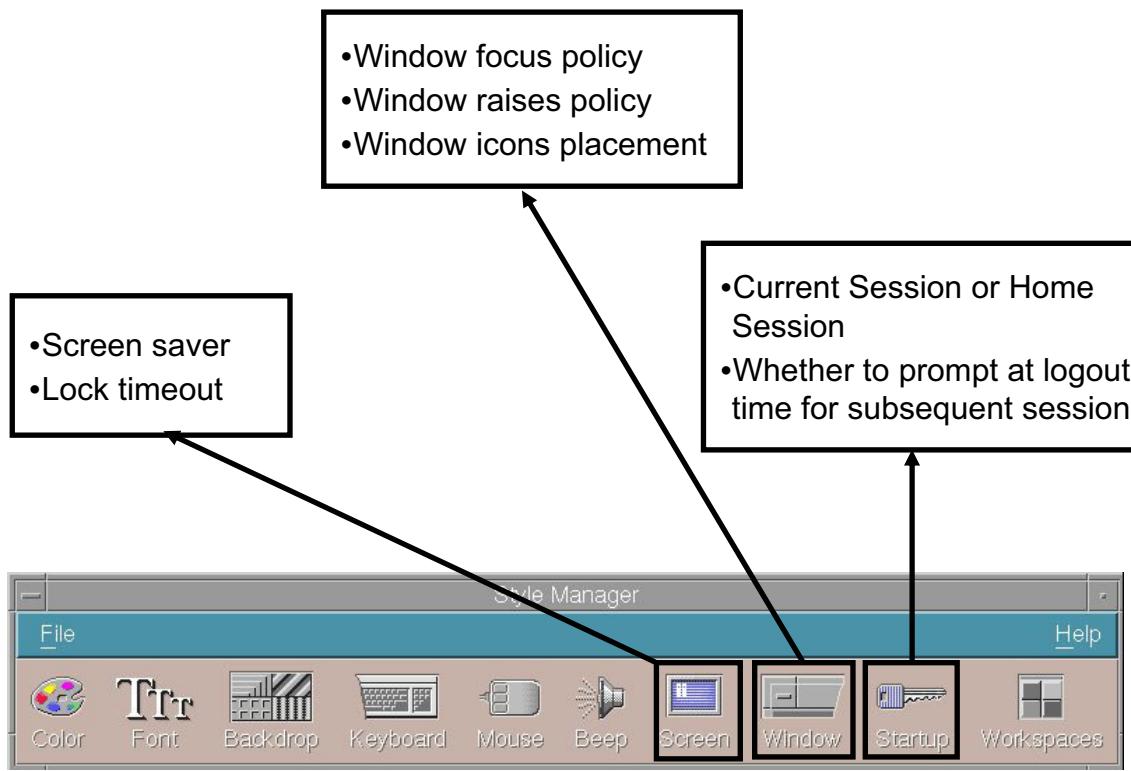
Details — Character repeat on the keyboard determines whether to repeat a character when its key is held down. The default is on.

Additional Information — With the mouse control it is possible to change the setting for the middle mouse button. You choose whether the middle mouse button is to be used for Transfer or Adjust operations for list and text. Transfer refers to the drag-and-drop action and Adjust is known as extend.

With the Transfer setting, you use mouse button two to drag and drop list or text items. With the Adjust setting, you extend list selections in a multiple-select list or extend the text selection in text fields with mouse button two and you drag and drop list and text items with mouse button one.

Transition Statement — Lastly, in terms of the Style Manager we will look at customizing window, screen, and startup.

Style Manager - Window, Screen, and Startup



© Copyright IBM Corporation 2008

Figure D-8. Style Manager - Window, Screen, and Startup

AU1310.0

Notes:

More customization options

It is possible to customize how a window will acquire focus, whether the window will raise when it receives focus, and where window icons are placed.

The screen saver dialog allows you to customize the screen saver pattern, such as a swarm of bees or fireworks. This is important to prevent bright colors from burning into the picture tube. It is also possible to set a screen saver lock that will be invoked after a set period of time (you choose in minutes).

Whenever you are logged in to CDE, you are working in a *current session*. By default, when you log out, the desktop saves your current session and restores it the next time you log in. You can specify that you prefer to log into your *home session* instead of your *current session*. It is also possible for CDE to ask you, at logout, which type of session you want to access the next time you log in.

Instructor Notes:

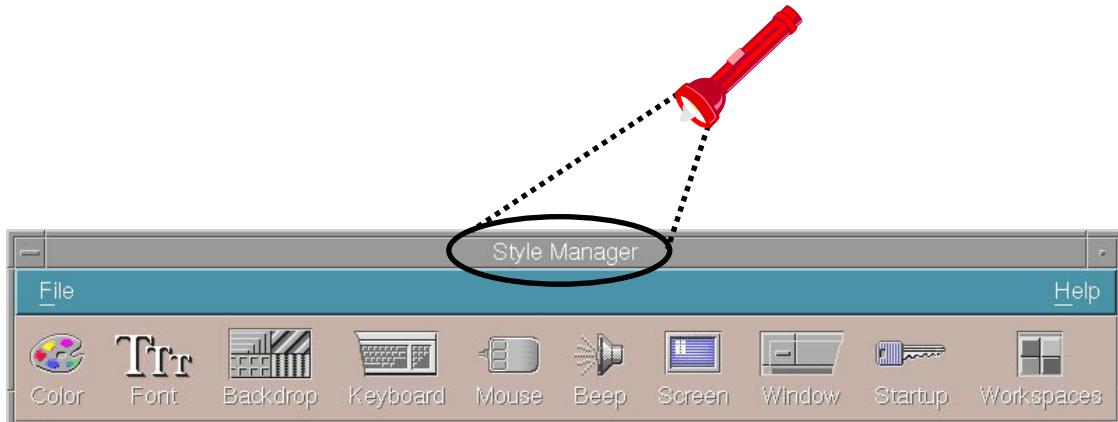
Purpose — Explain the window, screen, and startup customization options.

Details — There is also one more item on the Style Manager that is not on the visual, the Workspace control. This control is used to indicate if the workspace buttons should be shown on the Front Panel. The default is to show the workspace buttons.

Additional Information —

Transition Statement — Before we switch to the front panel, let's do an activity.

Activity: Review Style Manager



© Copyright IBM Corporation 2008

Figure D-9. Activity: Review Style Manager

AU1310.0

Notes:

Activity

- 1. Log in to the system.
- 2. Start the *Style Manager*.
- 3. Select a *larger font size* and start a new `dtterm` window afterwards.
- 4. Change the *backdrop* in your current workspace.
- 5. Change the *window behavior*: Point in window to make it active.
- 6. Specify a new *screen saver*.

Instructor Notes:

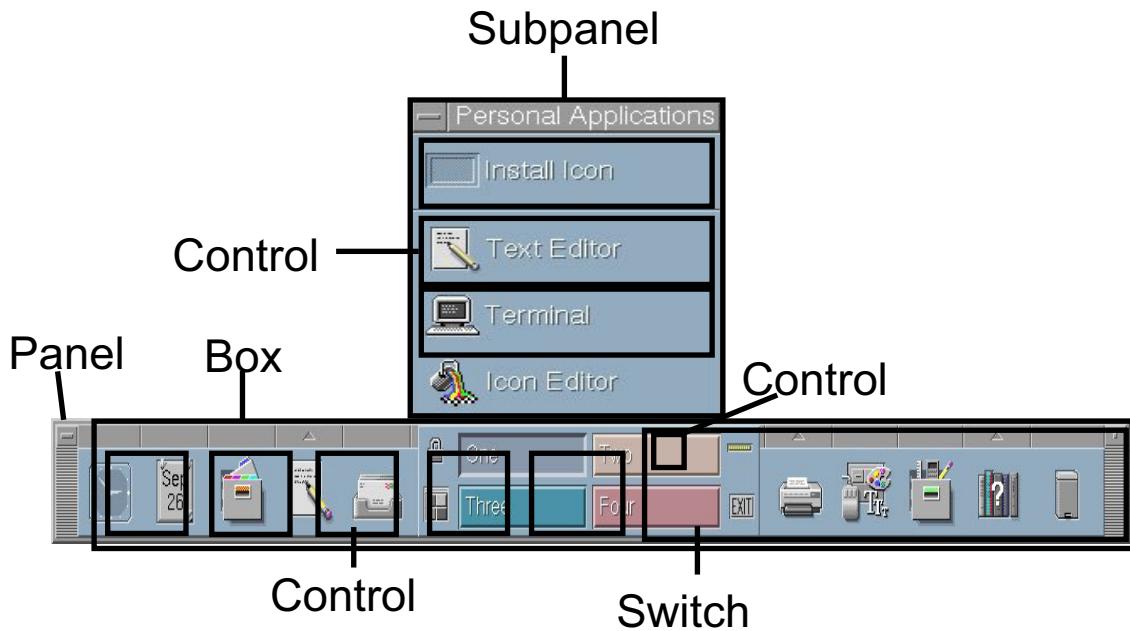
Purpose — Processing activity to review the Style Manager.

Details — Give the students some time to work with the Style Manager.

Additional Information —

Transition Statement — Let's introduce how to change the Front Panel.

General Structure of a Front Panel



© Copyright IBM Corporation 2008

Figure D-10. General Structure of a Front Panel

AU1310.0

Notes:

The CDE Front Panel

The Front Panel is built using a hierarchy of constructs, or components. Components can be containers for other components, or controls for user actions. Containers must be nested following specific rules.

There can be only one Front Panel/Main Panel. The Main Panel can contain only boxes (one by default, but can be customized to contain more). More than one box in the Main Panel results in a multirow Front Panel.

Front Panel components

Boxes are the horizontal containers for controls, the Workspace Switch Area, and the Subpanel Access Areas. There must be at least one box in the Main Panel.

There can be only one *Workspace Switch Area* in the Front Panel, contained in any of the existing boxes. It can contain workspace switch buttons and controls.

Subpanels can be attached to a control within a box and they contain additional controls. There can be only one subpanel per box-contained control.

The *controls* are the basic building block for all front panels. They allow starting of applications, but can also be used to display certain conditions (mail arrived). Controls are embedded in a box or in a subpanel.

It is important that this hierarchy and associated rules be followed when customizing the Front Panel. Broken rules result in unpredictable results.

Instructor Notes:

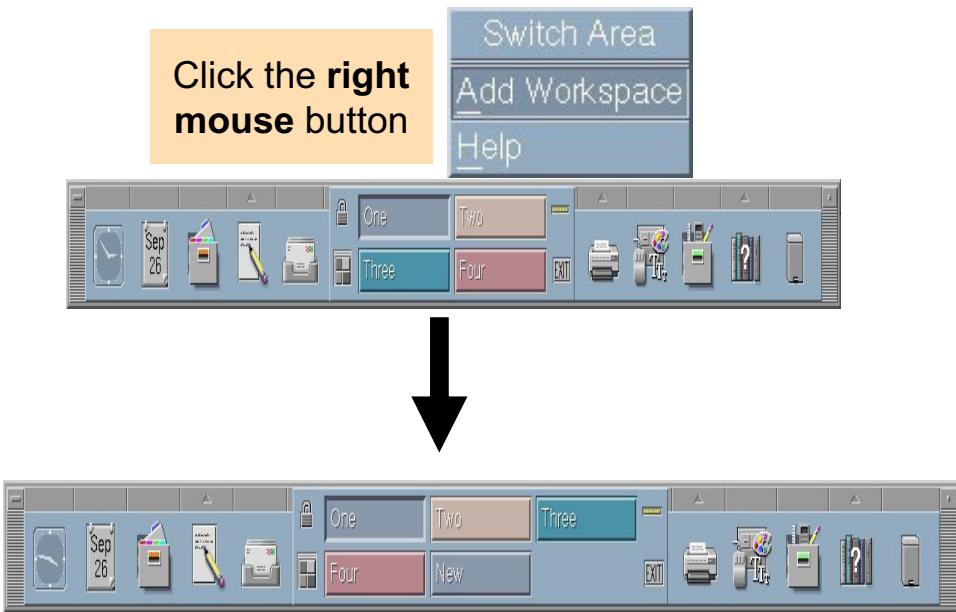
Purpose — Further discuss the hierarchy and rules associated with any customization of the Front Panel.

Details — Review the hierarchy and rules outlined in the student notes. Care must be taken when altering the Front Panel so that the rules are not broken. Interactive customization will probably be safe, with little opportunity to break the rules. However, more advanced users, who choose to modify the CDE customization files have to be very aware of the Front Panel hierarchy and rules as there is a greater chance of error. If errors are numerous, it is likely that the Front Panel will not be displayed at all.

Additional Information —

Transition Statement — Let's show how to add a new workspace.

Creating a New Workspace



© Copyright IBM Corporation 2008

Figure D-11. Creating a New Workspace

AU1310.0

Notes:

New workspace

Creating a new workspace is very simple. Use the right mouse button to click either the Workspace Switch area, or one of the Workspace Switch buttons. Then, click **Add Workspace** on the subpanel that appears. The new workspace is created with a name of New.

At logout, the new resource values will be written into the file:

\$HOME/.dt/sessions/current/dt.resources

and the old setting copied into

\$HOME/.dt/sessions/current.old/dt.resources.

You will find a new resource: `Dtwm*0*ws4*title: New` in the file. ws4 indicates the fifth workspace, since the count starts with ws0, and a modified resource:
`Dtwm*0*workspaceCount: 5`

Instructor Notes:

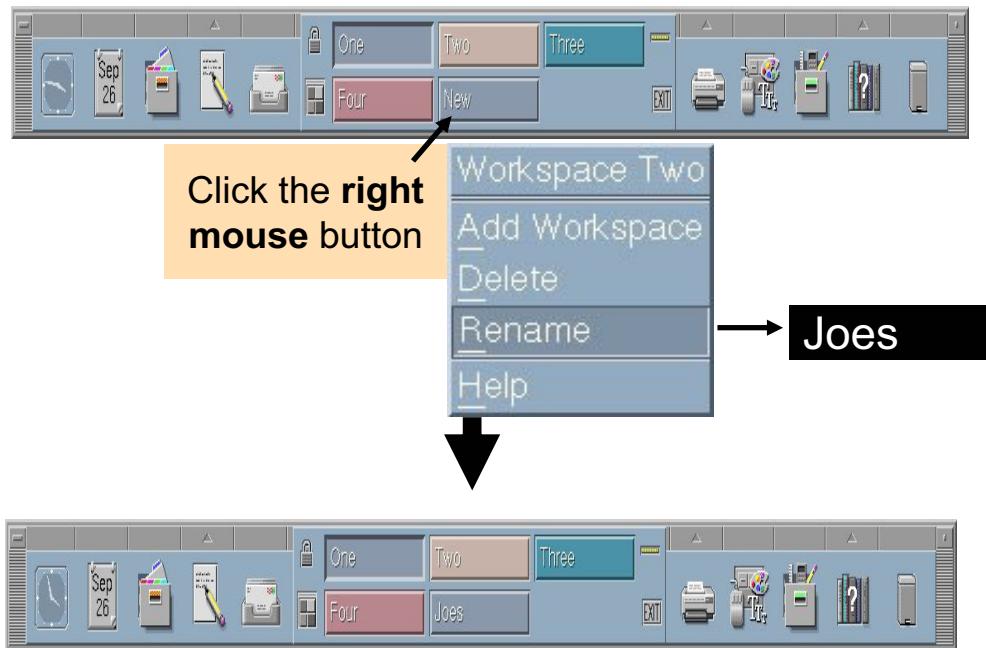
Purpose — Discuss how a new workspace can be created.

Details — The student notes describe how a new workspace can be easily created. This new workspace information will be saved in a file in the user's \$HOME directory so it will be available at the next login.

Additional Information —

Transition Statement — The new workspace will have a name of New. Most users will probably want to modify this name. We will discuss how to do this next.

Changing a Workspace Name



© Copyright IBM Corporation 2008

Figure D-12. Changing a Workspace Name

AU1310.0

Notes:

Changing the workspace name

To interactively change the name of a workspace, use the right mouse button to click the Workspace Switch button for the workspace whose name you want to change. A pop-up menu will be displayed. Click **Rename** and then type in the new name for the workspace. Press **Enter** once the new name is entered.

Alternately, click the Front Panel button for the workspace whose name you want to change. That workspace is displayed. Click the workspace's Front Panel button again. The button becomes a text field where you can type in the new name for the workspace.

Once the change has been made and you log out, the change is permanently recorded in the file:

\$HOME/.dt/sessions/current/dt.resources

and the old setting copied into

\$HOME/.dt/sessions/current.old/dt.resources.

The new resource will look something like:

Dtwm*0*ws0*title: Newname

Note that the workspace count starts with 0, so the first workspace is shown as ws0.

Instructor Notes:

Purpose — Explain how the workspace names can be changed.

Details — The student notes outline two ways to dynamically change the name of a workspace. Both methods are simple.

With the second approach, the new resources will look something like

```
Dtwm*0*ws0*title: Newname1  
Dtwm*0*ws1*title: Newname2  
Dtwm*0*ws2*title: Newname3  
Dtwm*0*ws3*title: Newname4
```

Additional Information — You may also manually change the workspace name:

- `xrdb -e $HOME/CDEres`
- `vi $HOME/CDEres`
- Modify the value of the resource `Dtwm*0*wsN*title` to the new name you want to use for the workspace. Remember the count starts from 0, so you're changing the (N+1)th workspace.
- Save the file.
- Restart the Workspace Manager from the root menu.

Transition Statement — We will next look at how we can dynamically add or delete a subpanel.

Dynamic Creation or Deletion of a Subpanel

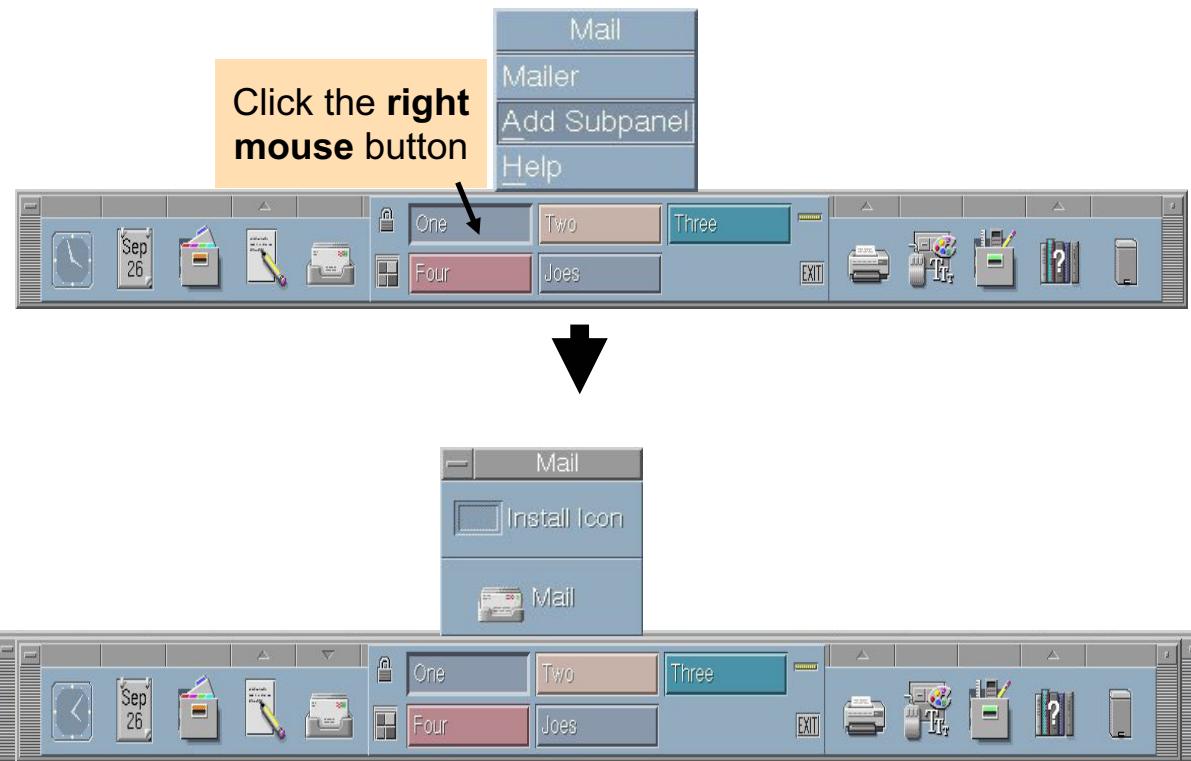


Figure D-13. Dynamic Creation or Deletion of a Subpanel

AU1310.0

Notes:

Introduction

Subpanels can be created only for controls directly contained in a box.

Adding a subpanel

To dynamically add a subpanel to a control, point to the control and press the right mouse button. On the pop-up menu that appears, choose Add Subpanel. This will add an arrow above the control. Press the arrow to view the subpanel. Additional items can be added to the subpanel as we will discuss shortly.

Creating a subpanel creates a new file in your personal environment. The file is placed in **\$HOME/.dt/types/fp_dynamic**. Its name depends on the name of the control the subpanel is attached to. For instance, when you create a subpanel for the Mail control, the subpanel description file name is **Mail1.fp**.

Since the only requirement for Front Panel description file names is that they end with the **.fp** extension, you are free to rename the files to any name you want.

Deleting a subpanel

To delete a subpanel, point to the control and press the right mouse button. On the pop-up menu that appears, choose **Delete Subpanel**. The file previously created will be removed.

Instructor Notes:

Purpose — Discuss how to dynamically add and delete a subpanel on a Front Panel control.

Details — To either add or delete a subpanel on a Front Panel control, point to the control and press the right mouse button. The pop-up menu allows subpanels to be added or deleted.

On the visual, the example shows adding a subpanel to the Mail control on the Front Panel. Once the subpanel is added, an arrow will appear above the Mail control.

These actions will automatically cause files in your **\$HOME/.dt/types/fp_dynamic** directory to be altered. Most users do not need to be familiar with the files that CDE updates, but some may want to know where this information is stored.

During the machine exercise, students will add a subpanel to the Style Manager control. By default, it does not have a subpanel.

Additional Information —

Transition Statement — We will next discuss how to add or remove a control on a subpanel.

Adding a Control to a Subpanel

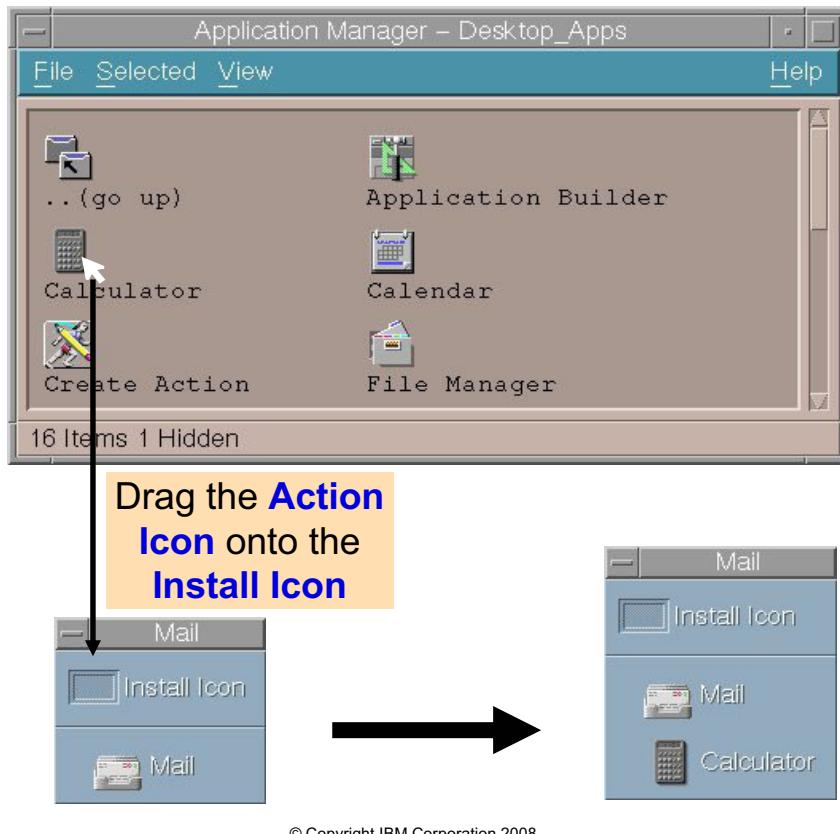


Figure D-14. Adding a Control to a Subpanel

AU1310.0

Notes:

Modifying subpanels

A control can be dynamically added to a subpanel by dropping an action's icon onto the *Install Icon* item on the subpanel.

Adding a control in a subpanel creates a new file in your personal environment. The file is placed in **\$HOME/.dt/types/fp_dynamic**. Its name depends on the name of the control added. For instance, if you drop the calculator icon into a subpanel, the control description file name will be **Dtcalc1.fp**. If you drop an **aixterm** icon into a subpanel, the control description file will be **Aixterm1.fp**.

To *delete* a control from a subpanel, point to the control and press the right mouse button on the subpanel to view the subpanel's pop-up window. Select the **Delete** option to delete the control. If all the controls on a subpanel are removed, the arrow above the Front Panel control will also disappear.

Instructor Notes:

Purpose — Briefly explain how a user can add and delete a control from a subpanel.

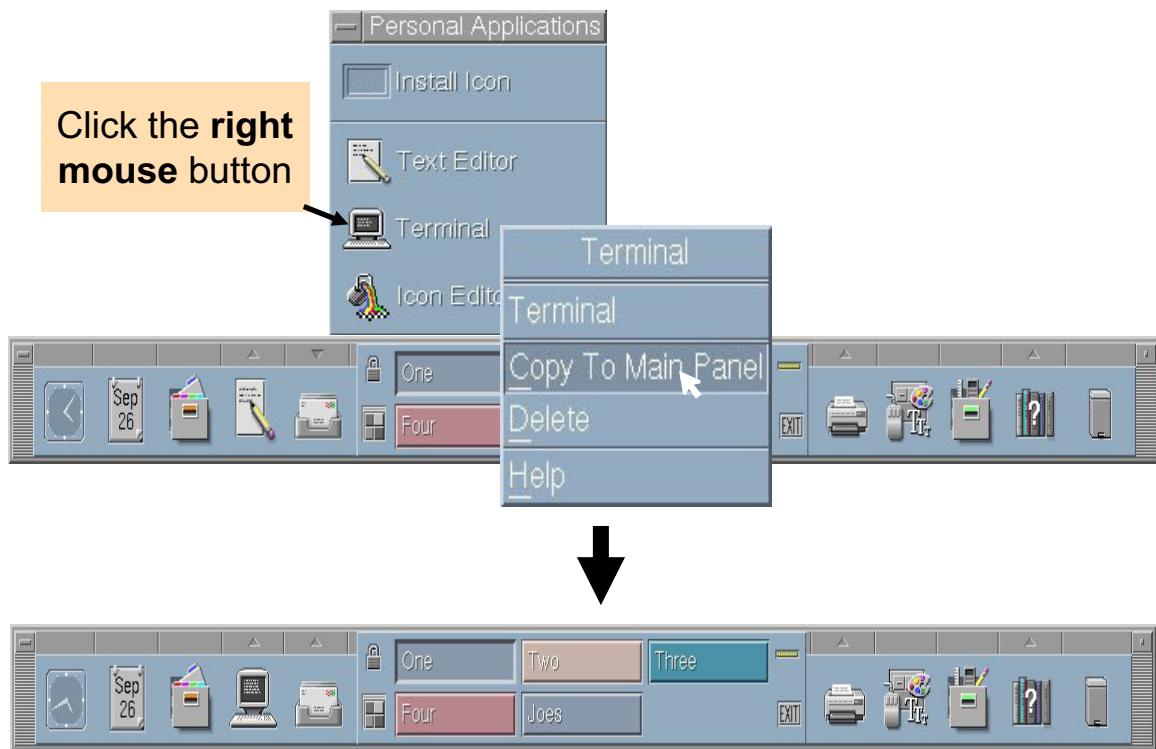
Details — The student notes describe how a user can easily add and remove a control from a subpanel. Students will do this during the machine exercise. They will create a subpanel for the Style Manager (it currently does not have a subpanel). Then, they will add two controls to the subpanel: the Icon Editor and an [aixterm](#). They will then remove the Icon Editor from the subpanel.

It is possible for the root user to deny the ability for a user to perform these functions. This would need to be coded in the description file, using the `CONTROL_INSTALL` field).

Additional Information —

Transition Statement — Next, we will look at how a subpanel control can be copied to the Front Panel.

Copy a Subpanel Control to the Main Panel



© Copyright IBM Corporation 2008

Figure D-15. Copy a Subpanel Control to the Main Panel

AU1310.0

Notes:

Modifying the Main Panel

By default, the first item in the subpanel is the one that is shown in the corresponding control in the Main Panel. Changing this default is quite easy. Just point to the control that you want to place on the Front Panel and press the right mouse button. On the pop-up menu that appears, choose **Copy to Main Panel**.

In our example, we are changing the control for Personal Applications from the Text Editor to the Terminal.

Instructor Notes:

Purpose — Discuss how a user can easily change the control on the Front Panel.

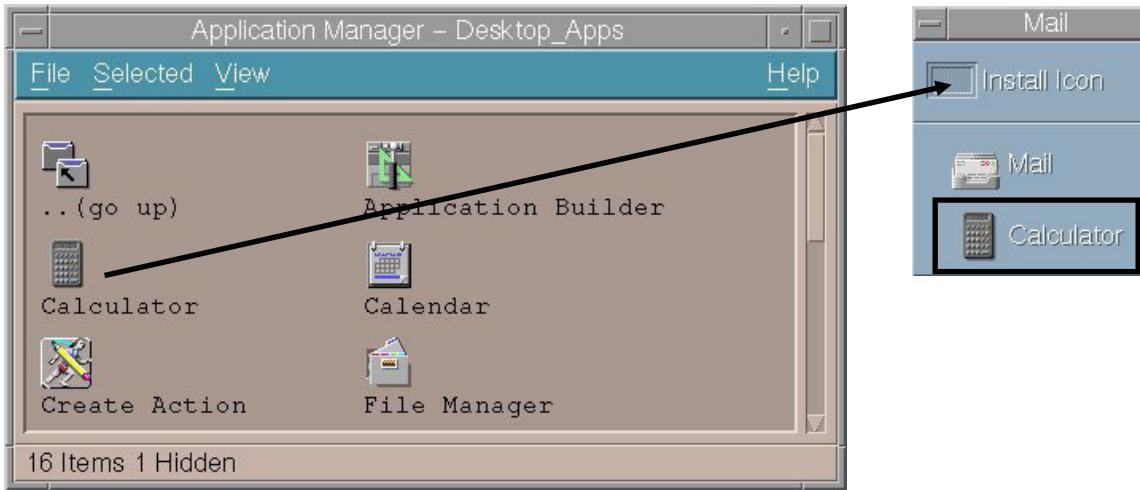
Details — In our example, we have decided that the Personal Applications control should show a Terminal rather than the Text Editor. The instructions on how to make this change are shown on both the visual as well as in the student notes. Since many users are more likely to need a `dtterm` rather than the CDE text editor, this may be a practical change to make.

Additional Information —

Transition Statement — Finally, let's show how to add controls to the front panel.

Adding Controls to the Front Panel (1 of 2)

1. Add the Control to a Subpanel:



2. Find out the name of the definition file for this control:

```
$ ls $HOME/.dt/types/fp_dynamic
Dtcalc1.fp
```

© Copyright IBM Corporation 2008

Figure D-16. Adding Controls to the Front Panel (1 of 2)

AU1310.0

Notes:

Adding controls to the Front Panel with the Application Manager

There are different ways that controls could be added to the Front Panel. The visual shows the easiest way, by using the Application Manager.

First the desired action is dropped into an existing subpanel, for example, the *mailer subpanel*. In the visual we drop the *Calculator Icon* onto the *Install Icon*.

This control which resides in the subpanel is described by a *definition file* in directory **\$HOME/.dt/types/fp_dynamic**. You must find out the name of the definition file, because you must work with this file. In the example, the *Calculator control* is described in the definition file **Dtcalc1.fp**.

Instructor Notes:

Purpose — Introduce how a control can be added to the Front Panel.

Details —

Additional Information —

Transition Statement — Let's show the second part.

Adding Controls to the Front Panel (2 of 2)

3. Copy the definition file to directory \$HOME/.dt/types

```
$ cp $HOME/.dt/types/fp_dynamic/Dtcalc1.fp $HOME/.dt/types/joe.fp
```

4. Anchor the Control in the Front Panel:

```
$ vi $HOME/.dt/types/joe.fp
```

```
CONTROL Dtcalc
{
    ...
    CONTAINER_TYPE      BOX
    CONTAINER_NAME      Top
    POSITION_HINTS      last
    ...
}
```

5. Restart the CDE

6. If you can not log in to the CDE, use Failsafe Session login

© Copyright IBM Corporation 2008

Figure D-17. Adding Controls to the Front Panel (2 of 2)

AU1310.0

Notes:

Adding Front Panel controls (continued)

The next thing you have to do is to copy the definition file from **\$HOME/.dt/types/fp_dynamic** into **\$HOME/.dt/types**. Use any desired name that you want, but add the suffix **.fp** to the file name. In the example the name **joe.fp** was used.

Next you must edit the copied definition file (**joe.fp**), to anchor the control in the Front Panel. This is easy: You just have to change the following lines, as shown in the visual:

```
CONTAINER_TYPE      BOX
CONTAINER_NAME      Top
POSITION_HINTS      last
```

After restarting the CDE, the *calculator control* is shown on the Front Panel. If you get any problems during the CDE login, use the *Failsafe Session* login.

Instructor Notes:

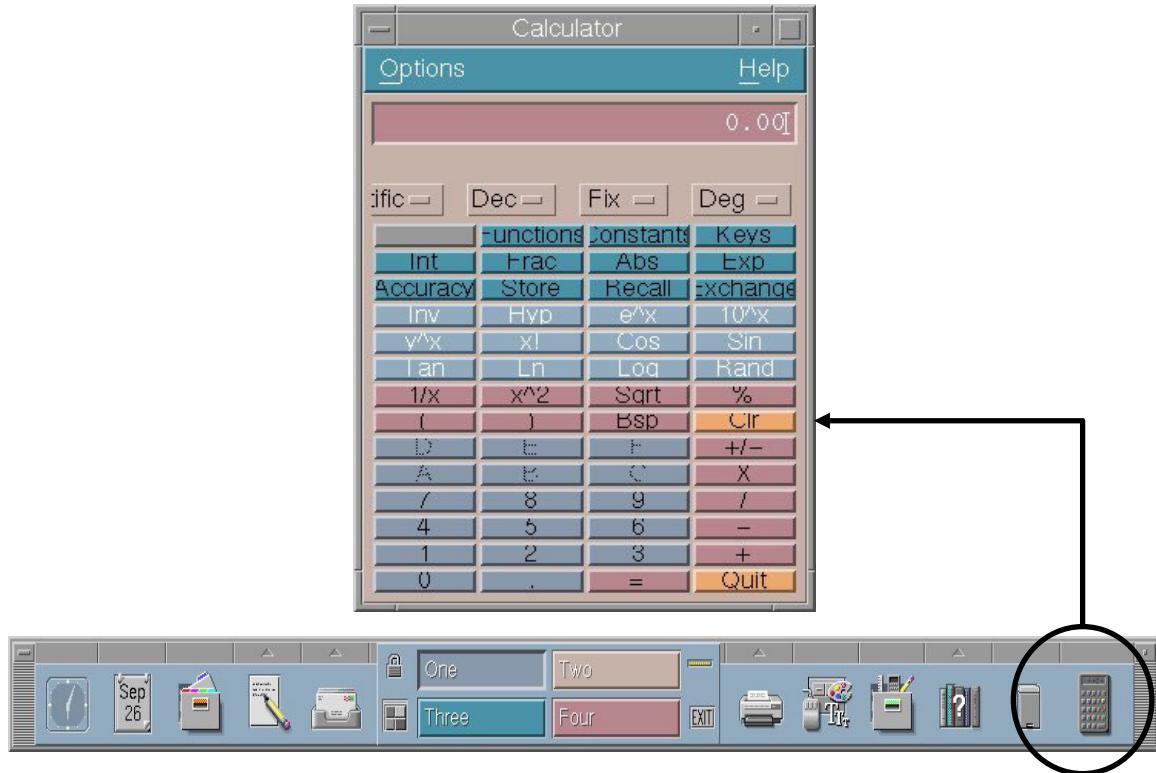
Purpose — Explain how to add a control to the Front Panel.

Details —

Additional Information —

Transition Statement — Let's show the extended Front Panel.

Extended Front Panel



© Copyright IBM Corporation 2008

Figure D-18. Extended Front Panel

AU1310.0

Notes:

The visual shows how the customized Front Panel looks after adding the *calculator control*.

Instructor Notes:

Purpose — Show the customized Front Panel.

Details —

Additional Information —

Transition Statement — Before going to lab, let's cover a few checkpoint questions.

Checkpoint

1. How do you customize the screen saver in your desktop environment ?
2. True or false: You can have more than four workspaces on the CDE front panel.
3. Describe how controls can be added to the CDE front panel.

© Copyright IBM Corporation 2008

Figure D-19. Checkpoint

AU1310.0

Notes:

Instructor Notes:

Purpose — To test the students' understanding of this unit's material.

Details —

Checkpoint Solutions

1. How do you customize the screen saver in your desktop environment ?
By using the Style Manager

2. True or false: You can have more than four workspaces on the CDE front panel.
True

3. Describe how controls can be added to the CDE front panel.
First add the control to a subpanel. Copy the definition file and anchor the control in the front panel.

© Copyright IBM Corporation 2008

Additional Information —

Transition Statement — Let's have a lab.

Exercise: Customizing CDE



© Copyright IBM Corporation 2008

Figure D-20. Exercise: Customizing CDE

AU1310.0

Notes:

After completing this lab, you will be able to:

- Customize the CDE using the *Style Manager*
- Customize the *Front Panel*

Instructor Notes:

Purpose — To introduce the last lab.

Details — Describe what students will learn.

Additional Information —

Transition Statement — Let's summarize this unit's important points.

Unit Summary

- Use the **Style Manager** to interactively **customize CDE** for color, fonts, backdrop, mouse, keyboard, beep and window behavior, the screen saver and startup options.
- Interactively **customize the Front Panel** to add and rename workspace switches, add or delete a subpanel, add or delete a control in a subpanel, and copy a subpanel control to the Main Panel.
- To **add controls to the front panel**, add them first to a subpanel. Copy the definition file and anchor the control in the front panel.

© Copyright IBM Corporation 2008

Figure D-21. Unit Summary

AU1310.0

Notes:

Instructor Notes:

Purpose — To summarize this unit's key points.

Details —

Additional Information —

Transition Statement — End of class!

Glossary

A

access mode A matrix of protection information stored with each file specifying who may do what to a file. Three classes of users (owner, group, all others) are allowed or denied three levels of access (read, write, execute).

access permission See **access mode**.

access privilege See **access mode**.

address space The address space of a process is the range of addresses available to it for code and data. The relationship between real and perceived space depends on the system and support hardware.

AIX Advanced Interactive Executive. IBM's implementation of the UNIX Operating System.

AIX Family Definition IBM's definition for the common operating system environment for all members of the AIX family. The AIX Family Definition includes specifications for the AIX Base System, User Interface, Programming Interface, Communications Support, Distributed Processing, and Applications.

alias The command and process of assigning a new name to a command.

ANSI American National Standards Institute. A standards organization. The United States liaison to the International Standards Organization (ISO).

application program A program used to perform an application or part of an application.

argument An item of information following a command. It may, for example, modify the command or identify a file to be affected.

ASCII American Standard Code for Information Interchange. A collection of public domain character sets considered standard throughout the computer industry.

awk An interpreter, included in most UNIX operating systems, that performs sophisticated text pattern matching. In combination with shell scripts, awk can be used to prototype or implement applications far more quickly than traditional programming methods.

B

background (process) A process is "in the background" when it is running independently of the initiating terminal. It is specified by ending the ordinary command with an ampersand (&). The parent of the background process does not wait for its "death."

backup diskette A diskette containing information copied from another diskette. It is used in case the original information is unintentionally destroyed.

Berkeley Software Distribution Disseminating arm of the UNIX operating system community at the University of California at Berkeley; commonly

abbreviated "BSD." Complete versions of the UNIX operating system have been released by BSD for a number of years; the latest is numbered 4.3. The phrase "Berkeley extensions" refers to features and functions, such as the C shell, that originated or were refined at UC Berkeley and that are now considered a necessary part of any fully-configured version of the UNIX operating system.

bit bucket The AIX file "/dev/null" is a special file which will absorb all input written to it and return no data (null or end of file) when read.

block A group of records that is recorded or processed as a unit.

block device A device that transfers data in fixed size blocks. In AIX, normally 512 or 1024 bytes.

block special file An interface to a device capable of supporting a file system.

booting Starting the computer from scratch (power off or system reset).

break key The terminal key used to unequivocally interrupt the foreground process.

BSD Berkeley Software Distribution.

- BSD 2.x - PDP-11 Research
 - BSD 4.x - VAX Research
 - BSD 4.3 - Current popular VAX version of UNIX.
1. A word, number, symbol, or picture on the screen that can be selected. A button may represent a command, file, window, or value, for example.
 2. A key on a mouse that is used to select buttons on the display screen or to scroll the display image.

byte The amount of storage required to represent one character; a byte is 8 bits.

C

C The programming language in which the UNIX operating system and most UNIX application programs are written. The portability attributed to UNIX operating systems is largely due to the fact that C, unlike other higher level languages, permits programmers to write systems-level code that will work on any computer with a standard C compiler.

change mode The **chmod** command will change the access rights to your own files only, for yourself, your group or all others.

character I/O The transfer of data byte by byte; normally used with slower, low-volume devices such as terminals or printers.

character special file An interface to devices not capable of supporting a file system; a byte-oriented device.

child The process emerging from a fork command with a zero return code, as distinguished from the parent which gets the process ID of the child.

client User of a network service. In the client/server model, network elements are defined as either using (client) or providing (server) network resources.

command A request to perform an operation or run a program. When parameters, arguments, flags, or other operands are associated with a command, the resulting character string is a single command.

command file A data file containing shell commands. See **shell file**, or **shell script**.

command interpreter The part of the operating system that translates your commands into instructions that the operating system understands.

concatenate The process of forming one character string or file from several. The degenerate case is one file from one file just to display the result using the **cat** command.

console The only terminal known explicitly to the Kernel. It is used during booting and it is the destination of serious system messages.

context The hardware environment of a process, including:

- CPU registers
- Program address
- Stack
- I/O status

The entire context must be saved during a process swap.

control character Codes formed by pressing and holding the **control** key and then some other key; used to form special functions like **End Of File**.

control-d See **eof** character.

cooked input Data from a character device from which backspace, line kill, and interrupt characters have been removed (processed). See **raw input**.

current directory The currently active directory. When you specify a file name without specifying a directory, the system assumes that the file is in your current directory.

current subtree Files or directories attached to the current directory.

curses A C subroutine library providing flexible screen handling. See **Termlib** and **Termcap**.

cursor A movable symbol (such as an underline) on a display, usually used to indicate to the operator where to type the next character.

customize To describe (to the system) the devices, programs, users, and user defaults for a particular data processing system.

D

DASD Direct Access Storage Device. IBM's term for a hard disk.

device driver. A program that operates a specific device, such as a printer, disk drive, or display.

device special file A file which passes data directly to/from the device.

directory A type of file containing the names and controlling information for other files or other directories.

directory pathname The complete and unique external description of a file giving the sequence of connection from the root directory to the specified directory or file.

diskette A thin, flexible magnetic plate that is permanently sealed in a protective cover. It can be used to store information copied from the disk.

diskette drive The mechanism used to read and write information on diskettes.

display device An output unit that gives a visual representation of data.

display screen The part of the display device that displays information visually.

E

echo To simply report a stream of characters, either as a message to the operator or a debugging tool to see what the file name generation process is doing.

editor A program used to enter and modify programs, text, and other types of documents.

environment A collection of values passed either to a C program or a shell script file inherited from the invoking process.

escape The backslash "\ character specifies that the single next character in a command is ordinary text without special meaning.

Ethernet A baseband protocol, invented by the XEROX Corporation, in common use as the local area network for UNIX operating systems interconnected via TCP/IP.

event One of the previous lines of input from the terminal. Events are stored in the (Berkeley) History file.

event identifier A code used to identify a specific event.

execution permission For a file, the permission to execute (run) code in the file. A text file must have execute permission to be a shell script. For a directory, the permission to search the directory.

F

field A contiguous group of characters delimited by blanks. A field is the normal unit of text processed by text processes like sort.

field separator The character used to separate one field from the next; normally a blank or tab.

FIFO First In First Out. In AIX, a FIFO is a permanent, named pipe which allows two unrelated processes to communicate. Only related processes can use normal pipes.

file A collection of related data that is stored and retrieved by an assigned name. In AIX, files are grouped by directories.

file index Sixty-four bytes of information describing a file. Information such as the type and size of the file and the location on the physical device on which the data in the file is stored is kept in the file index. This index is the same as the AIX Operating System i-node.

filename expansion or generation A procedure used by the shell to generate a set of filenames based on a specification using metacharacters, which define a set of textual substitutions.

file system The collection of files and file management structures on a physical or logical mass storage device, such as a diskette or minidisk.

filter Data-manipulation commands (which, in UNIX operating systems, amount to small programs) that take input from one process and perform an operation yielding new output. Filters include editors, pattern-searchers, and commands that sort or differentiate files, among others.

fixed disk A storage device made of one or more flat, circular plates with magnetic surfaces on which information can be stored.

fixed disk drive The mechanism used to read and write information on a fixed disk.

flag See **Options**.

foreground (process) An AIX process which interacts with the terminal. Its invocation is not followed by an ampersand.

formatting The act of arranging text in a form suitable for reading. The publishing equivalent to compiling a program.

fsck A utility to check and repair a damaged file structure. This normally results from a power failure or hardware malfunction. It looks for blocks not assigned to a file or the free list and puts them in the free list. (The use of blocks not pointed at cannot be identified.)

free list The set of all blocks not assigned to a file.

full path name The name of any directory or file expressed as a string of directories and files beginning with the root directory.

G

gateway A device that acts as a connector between two physically separate networks. It has interfaces to more than one network and can translate the packets of one network to another, possibly dissimilar network.

global Applying to all entities of a set. For example:

- A global search - look everywhere
- A global replace - replace all occurrences
- A global symbol - defined everywhere

grep An AIX command which searches for strings specified by a regular expression. (Global Regular Expression and Print.)

group A collection of AIX users who share a set of files. Members of the group have access privileges exceeding those of other users.

H

hardware The equipment, as opposed to the programming, of a system.

header A record at the beginning of the file specifying internal details about the file.

heterogeneous Descriptor applied to networks composed of products from multiple vendors.

hierarchy A system of objects in which each object belongs to a group. Groups belong to other groups. Only the head does not belong to another group. In AIX this object is called the Root Directory.

highlight To emphasize an area on the display screen by any of several methods, such as brightening the area or reversing the color of characters within the area.

history A list of recently executed commands.

- A directory associated with an individual user.
- Your current directory on login or after issuing the **cd** command with no argument.

homogeneous Descriptor applied to networks composed of products from a single vendor.

hypertext Term for online interactive documentation of computer software; to be included with AIX.

I

IEEE Institute of Electrical and Electronics Engineers. A professional society active in standards work, the IEEE is the official body for work on the POSIX (Portable Operating System for Computer Environments) open system interface definition.

index See **file index**.

indirect block A file element which points at data sectors or other indirect blocks.

init The initialization process of AIX. The ancestor of all processes.

initial program load The process of loading the system programs and preparing the system to run jobs.

i-node A collection of logical information about a file including owner, mode, type, and location.

i number The internal index or identification of an i-node.

input field An area into which you can type data.

input redirection The accessing of input data from other than standard input (the keyboard or a pipe).

interoperability The ability of different kinds of computers to work well together.

interpreter A program which interprets program statements directly from a text (or equivalent) file. Distinguished from a compiler which creates computer instructions for later direct execution.

interrupt A signal that the operating system must reevaluate its selection of which process should be running. Usually to service I/O devices but also to signal from one process to another.

IP Internet Protocol.

ipl See **initial program load**.

ISO International Standards Organization. A United Nations agency that provides for creation and administration of worldwide standards.

J

job A collection of activities.

job number An identifying number for a collection of processes devolving from a terminal command.

K

kernel The part of an operating system that contains programs that control how the computer does its work, such as input/output, management and control of hardware, and the scheduling of user tasks.

keyboard An input device consisting of various keys allowing the user to input data, control cursor and pointer locations, and to control the user/work station dialog.

kill To prematurely terminate a process.

kill character The character which erases an entire line (usually @).

L

LAN Local Area Network. A facility, usually a combination of wiring, transducers, adapter boards, and software protocols, which interconnects workstations and other computers located within a department, building, or neighborhood. Token Ring and Ethernet are local area network products.

libc A basic set of C callable routines.

library In UNIX operating systems, a collection of existing subroutines that allows programmers to make use of work already done by other programmers. UNIX operating systems often include separate libraries for communications, window management, string handling, math, and so on.

line editor An editor which processes one line at a time by the issuing of a command. Usually associated with sequential only terminals such as a teletype.

link An entry in an AIX directory specifying a data file or directory and its name. Note that files and directories are named solely by virtue of links. A name is not an intrinsic property of a file. A file is uniquely identified only by a system generated identification number.

lint A program for removing fuzz from C code. Stricter than most compilers. Helps former Pascal programmers sleep at night.

Local Area Network (LAN) A facility, usually a combination of wiring, transducers, adapter boards,

and software protocols, which interconnects workstations and other computers located within a department, building, or neighborhood. Token Ring and Ethernet are local area network products.

login Identifying oneself to the system to gain access.

login directory See **home directory**.

login name The name by which a user is identified to the system.

logout Informing the system that you are through using it.

M

mail The process of sending or receiving an electronically delivered message within an AIX system. The message or data so delivered.

make Programming tool included in most UNIX operating systems that helps make a new program out of a collection of existing subroutines and utilities, by controlling the order in which those programs are linked, compiled, and executed.

map The process of reassigning the meaning of a terminal key. In general, the process of reassigning the meaning of any key.

memory Storage on electronic memory such as random access memory, read only memory, or registers. See **storage**.

message Information displayed about an error or system condition that may or may not require a user response.

motd Message of the day. The login billboard message.

Motif The graphical user interface for OSF, incorporating the X Window System. Behavior of this interface is compatible with the IBM/Microsoft Presentation Manager user interface for OS/2. Also called OSF/Motif.

mount A logical (that is, not physical) attachment of one file directory to another. Remote mounting allows files and directories that reside on physically separate computer systems to be attached to a local system.

mouse A device that allows you to select objects and scroll the display screen by means of buttons.

move Relinking a file or directory to a different or additional directory. The data (if any) is not moved, only the links.

multiprogramming Allocation of computer resources among many programs. Used to allow many users to operate simultaneously and to keep the system busy during delays occasioned by I/O mechanical operations.

multitasking Capability of performing two or more computing tasks, such as interactive editing and complex numeric calculations, at the same time. AIX and OS/2 are multitasking operating systems; DOS, in contrast, is a single-tasking system.

multiuser A computer system which allows many people to run programs simultaneously using multiprogramming techniques.

N

named pipe See FIFO.

Network File System (NFS) A program developed by SUN Microsystems, Inc. for sharing files among systems connected via TCP/IP. IBM's AIX, VM, and MVS operating systems support NFS.

NFS See Network File System.

NIST National Institute of Science and Technology (formerly the National Bureau of Standards).

node An element within a communication network.

- Computer
- Terminal
- Control Unit

null A term denoting emptiness or nonexistence.

null device A device used to obtain empty files or dispose of unwanted data.

null string A character string containing zero characters.

O

object-oriented programming Method of programming in which sections of program code and data are represented, used, and edited in the form of objects, such as graphical elements, window components, and so forth, rather than as strict computer code. Through object-oriented programming techniques, toolkits can be designed that make programming much easier. Examples of object-oriented programming languages include Parcplace Systems, Inc.'s Smalltalk-80, AT&T's C++, and Stepstone Inc.'s Objective-C.

OEM original equipment manufacturer. In the context of AIX, OEM systems refer to the processors of a heterogeneous computer network that are not made or provided by IBM.

Open Software Foundation (OSF) A non-profit consortium of private companies, universities, and research institutions formed to conduct open technological evaluations of available components of UNIX operating systems, for the purpose of assembling selected elements into a complete version of the UNIX operating system available to those who wish to license it. IBM is a founding sponsor and member of OSF.

operating system The programs and procedures designed to cause a computer to function, enabling the user to interact with the system.

option A command argument used to specify the details of an operation. In AIX an option is normally preceded by a hyphen.

ordinary file Files containing text, programs, or other data, but not directories.

OSF See Open Software Foundation.

output redirection Passing a program's standard output to a file.

owner The person who created the file or his subsequent designee.

P

packet switching The transmission of data in small, discrete switching packets rather than in streams, for the purpose of making more efficient use of the physical data channels. Employed in some UNIX system communications.

page To move forward or backward on a screen full of data through a file usually referring to an editor function.

parallel processing A computing strategy in which a single large task is separated into parts, each of which then runs in parallel on separate processors.

parent The process emerging from a Fork with a non-zero return code (the process ID of the child process). A directory which points at a specified directory.

password A secret character string used to verify user identification during login.

PATH A variable which specifies which directories are to be searched for programs and shell files.

path name A complete file name specifying all directories leading to that file.

pattern-matching character Special characters such as * or ? that can be used in a file specification to match one or more characters. For example, placing a ? in a file specification means that any character can be in that position.

permission The composite of all modes associated with a file.

pipes UNIX operating system routines that connect the standard output of one process with the standard input of another process. Pipes are central to the function of UNIX operating systems, which generally consist of numerous small programs linked together into larger routines by pipes. The piping of the list directory command to the word count command is **ls | wc**. The passing of data by a pipe does not (necessarily) involve a file. When the first program generates enough data for the second program to process, it is suspended and the second program runs. When the second program runs out of data it is suspended and the first one runs.

pipe fitting Connecting two programs with a pipe.

pipeline A sequence of programs or commands connected with pipes.

portability Desirable feature of computer systems and applications, referring to users' freedom to run application programs on computers from many vendors without rewriting the program's code. Also known as applications portability, machine-independence, and hardware-independence; often cited as a cause of the recent surge in popularity of UNIX operating systems.

port A physical I/O interface into a computer.

POSIX Portable Operating Systems for Computer Environments. A set of open standards for an operating system environment being developed under the aegis of the IEEE.

preprocessor The macro generator preceding the C compiler.

process A unit of activity known to the AIX system, usually a program.

process 0 (zero) The scheduler. Started by the boot and permanent. See **init**.

process ID A unique number (at any given time) identifying a process to the system.

process status The process's current activity.

- Non-existent
- Sleeping
- Waiting
- Running
- Intermediate
- Terminated
- Stopped

profile A file in the user's home directory which is executed at login to customize the environment. The name is **.profile**.

prompt A displayed request for information or operator action.

protection The opposite of permission, denying access to a file.

Q

quotation Temporarily canceling the meaning of a metacharacter to be used as a ordinary text character. A backslash (\) "quotes" the next character only.

R

raw I/O I/O conducted at a physical level.

read permission Allows reading (not execution or writing) of a file.

recursive A recursive program calls itself or is called by a subroutine which it calls.

redirection The use of other than standard input (keyboard or pipe output) or standard output (terminal display or pipe). Usually a file.

regular expression An expression which specifies a set of character strings using metacharacters.

relative path name The name of a directory or file expressed as a sequence of directories followed by a file name, beginning from the current directory.

RISC Reduced Instruction Set Computer. A class of computer architectures, pioneered by IBM's John Cocke, that improves price performance by minimizing the number and complexity of the operations required in the instruction set of a computer. In this class of architecture, advanced compiler technology is used to provide operations,

such as multiplication, that are infrequently used in practice.

root directory The directory that contains all other directories in the file system.

S

scalability Desirable feature of computer systems and applications. Refers to the capability to use the same environment on many classes of computers, from personal computers to supercomputers, to accommodate growth or divergent environments, without rewriting code or losing functionality.

SCCS Source Code Control System. A set of programs for maintaining multiple versions of a file using only edit commands to specify alternate versions.

scope The field of an operation or definition. Global scope means all objects in a set. Local scope means a restriction to a subset of the objects.

screen See **display screen**.

scroll To move information vertically or horizontally to bring into view information that is outside the display screen or pane boundaries.

search and replace The act of finding a match to a given character string and replacing each occurrence with some other string.

search string The pattern used for matching in a search operation.

sed Non-interactive stream editor used to do batch editing. Often used as a tool within shell scripts.

server A provider of a service in a computer network; for example, a mainframe computer with large storage capacity may play the role of database server for interactive terminals. See **client**.

setuid A permission which allows the access rights of a program owner to control the access to a file. The program can act as a filter for user data requests.

shell The outermost (user interface) layer of UNIX operating systems. Shell commands start and control other processes, such as editors and compilers; shells can be textual or visual. A series of system commands can be collected together into a shell script that executes like a batch (.BAT) file in DOS.

shell program A program consisting of a sequence of shell commands stored in an ordinary text file which has execution permission. It is invoked by simply naming the file as a shell command.

shell script See **shell program**.

single user (mode) A temporary mode used during booting of the AIX system.

signal A software generated interrupt to another process. See **kill**.

sockets Destination points for communication in many versions of the UNIX operating system, much as electrical sockets are destination points for electrical plugs. Sockets, associated primarily with 4.3 BSD, can be customized to facilitate

communication between separate processes or between UNIX operating systems.

software Programs.

special character See **metacharacter**.

special file A technique used to access I/O devices in which pseudo files are used as the interface for commands and data.

standard error The standard device at which errors are reported, normally the terminal. Error messages may be directed to a file.

standard input The source of data for a filter, which is by default obtained from the terminal, but which may be obtained from a file or the standard output of another filter through a pipe.

standard output The output of a filter which normally is by default directed to the terminal, but which may be sent to a file or the standard input of another filter through a pipe.

stdio A standard I/O package of C routines.

sticky bit A flag which keeps commonly used programs stick to the swapping disk for performance.

stopped job A job that has been halted temporarily by the user and which can be resumed at his command.

storage In contrast to memory, the saving of information on physical devices such as fixed disk or tape. See **memory**.

store To place information in memory or onto a diskette, fixed disk, or tape so that it is available for retrieval and updating.

streams Similar to sockets, streams are destination points for communications in UNIX operating systems. Associated primarily with UNIX System V, streams are considered by some to be more elegant than sockets, particularly for interprocess communication.

string A linear collection of characters treated as a unit.

subdirectory A directory which is subordinate to another directory.

subtree That portion of an AIX file system accessible from a given directory below the root.

suffix A character string attached to a file name that helps identify its file type.

superblock Primary information repository of a file system (location of i-nodes, free list, and so forth).

superuser The system administration; a user with unique privileges such as upgrading execution priority and write access to all files and directories.

superuser authority The unrestricted ability to access and modify any part of the operating system. This authority is associated with the user who manages the system.

SVID System V Interface Definition. An AT&T document defining the standard interfaces to be used by UNIX System V application programmers and users.

swap space (disk) That space on an I/O device used to store processes which have been swapping out to make room for other processes.

swapping The process of moving processes between main storage and the "swapping device," usually a disk.

symbolic debugger Program for debugging other programs at the source code level. Common symbolic debuggers include sdb, dbx, and xdbx.

sync A command which copies all modified blocks from RAM to the disk.

system The computer and its associated devices and programs.

system unit The part of the system that contains the processing unit, the disk drive and the disk, and the diskette drive.

System V AT&T's recent releases of its UNIX operating system are numbered as releases of UNIX System V.

T

TCP Transmission Control Protocol. A facility for the creation of reliable bytestreams (byte-by-byte, end-to-end transmission) on top of unreliable datagrams. The transmission layer of TCP/IP is used to interconnect applications, such as FTP, so that issues of re-transmission and blocking can be subordinated in a standard way. See **TCP/IP**.

TCP/IP Transmission Control Protocol/Internet Protocol. Pair of communications protocol considered defacto standard in UNIX operating system environments. IBM TCP/IP for VM and IBM TCP/IP for MVS are licensed programs that provide VM and MVS users with the capability of participating in networks using the TCP/IP protocol suite.

termcap A file containing the description of several hundred terminals. For use in determining communication protocol and available function.

termlib A set of C programs for using **termcap**.

tools Compact, well designed programs to perform specific tasks. More complex processes are performed by sequences of tools, often in the form of pipelines which avoid the need for temporary files.

two-digit display Two seven-segment light-emitting diodes (LEDs) on the operating panel used to track the progress of Power On Self Tests (POSTs).

U

UNIX Operating System A multiuser, multitasking interactive operating system created at AT&T Bell Laboratories that has been widely used and developed by universities, and that now is becoming increasingly popular in a wide range of commercial applications. See **kernel, shell, library, pipes, filters**.

user interface The component of the AIX Family Definition that describes common user interface

functions for the AIX PS/2, AIX/RT, and AIX/370 operating systems.

/usr/grp One of the oldest, and still active, user groups for the UNIX operating systems. IBM is a member of /usr/grp.

uucp A set of AIX utilities allowing:

- Autodial of remote systems
- Transfer of files
- Execution of commands on the remote system
- Reasonable security

V

vi Visual editor. A character editor with a very powerful collection of editing commands optimized for ASCII terminals; associated with BSD versions of the UNIX operating system.

visual editor An optional editor provided with AIX in which changes are made by modifying an image of the file on the screen, rather than through the exclusive use of commands.

W

wild card A metacharacter used to specify a set of replacement characters and thus a set of file names. For example * is any zero or more characters and ? is any one character.

window A rectangular area of the screen in which the dialog between you and a given application is displayed.

working directory The directory from which file searches are begun if a complete pathname is not specified. Controlled by the **cd** (change directory) command.

workstation A device that includes a keyboard from which an operator can send information to the system, and a display screen on which an operator can see the information sent to or received from the computer.

write Sending data to an I/O device.

write permission Permission to modify a file or directory.

X

X/Open An international consortium, including many suppliers of computer systems, concerned with the selection and adoption of open system standards for computing applications. IBM is a corporate sponsor of X/Open. See **Common Application Environment**.

X Windows IBM's implementation of the X Window System developed at the Massachusetts Institute of Technology with the support of IBM and DEC, that gives users windows into applications and processes not located only or specifically on their own console or computer system. X Windows is a powerful vehicle for distributing applications among users on heterogeneous networks.

Y

yacc "Yet Another Compiler Compiler." For producing new command interfaces.

Z

zeroeth argument The command name; the argument before the first.

IBM[®]