AIX 5L Basics

(Course Code AU13)

Student Notebook

ERC 9.0

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®	PS/2®
pSeries®	RISC System/6000®	RS/6000®

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

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

May 2005 Edition

Contents

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®	PS/2®
pSeries®	RISC System/6000®	RS/6000®

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

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

Course Description

AIX 5L Basics

Duration: 5 days

Purpose

This course enables students to perform everyday tasks using the AIX 5L operating system version 5.3.

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 and customize the AIXwindows environment
- Use and customize the Common Desktop Environment (CDE)

Contents

- Introduction to AIX
- Using the System
- AIX Documentation
- Files and Directories
- Using Files
- File Permissions
- Shell Basics
- Using Shell Variables
- The vi Editor
- Processes
- Customizing the User Environment
- AIX Utilities
- AIX Utilities, Part II
- Additional Shell Features
- AIXwindows Concepts
- Customizing AIXwindows
- Using the Common Desktop Environment (CDE)
- CDE User Customization

Curriculum relationship

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

Agenda

Day 1

Welcome
Unit 1 - Introduction to AIX
Unit 2 - Using the System
Exercise 1 - Using the System
Unit 3 - AIX Documentation
Exercise 2 - AIX Documentation
Unit 4 - Files and Directories
Exercise 3 - Files and Directories
Unit 5 - Using Files
Exercise 4 - Using Files

Day 2

Unit 6 - File Permissions
Exercise 5 - File Permissions
Unit 7 - The vi Editor
Exercise 6 - The vi Editor
Unit 8 - Shell Basics
Exercise 7 - Shell Basics
Unit 9 - Using Shell Variables
Exercise 8 - Using Shell Variables

Day 3

Unit 10 - Processes
Unit 11 - Controlling Processes
Exercise 9 - Controlling Processes
Unit 12 - Customizing the User Environment
Exercise 10 - Customizing the User Environment
Unit 13 - AIX Utilities
Exercise 11 - AIX Utilities (1)
Unit 13 - AIX Utilities (Continued)
Exercise 12 - AIX Utilities (2)
Unit 14 - AIX Utilities, Part II
Exercise 13 - AIX Utilities (3)

Day 4

Unit 14 - AIX Utilities, Part II (Continued)
Exercise 14 - AIX Utilities (4)
Unit 15 - Additional Shell Features
Exercise 15 - Additional Shell Features
Unit 16 - AIXwindows Concepts
Exercise 16 - Using AIXwindows
Unit 17 - Customizing AIXwindows
Exercise 17 - Customizing AIXwindows (1)

Day 5

Unit 17 - Customizing AIXwindows (Continued)
Exercise 18 - Customizing AIXwindows (2)
Unit 18 - Using Common Desktop Environment (CDE)
Exercise 19 - Using CDE
Unit 19 - CDE User Customization
Exercise 20 - Customizing CDE

Text highlighting

The following text highlighting conventions are used throughout this book:

Bold	Identifies file names, file paths, directories, user names and principals.
Italics	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

What This Unit Is About

This unit is an introduction to the course AIX 5L 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

Figure 1-1. Unit Objectives AU139.0

Notes:

AIX Operating System

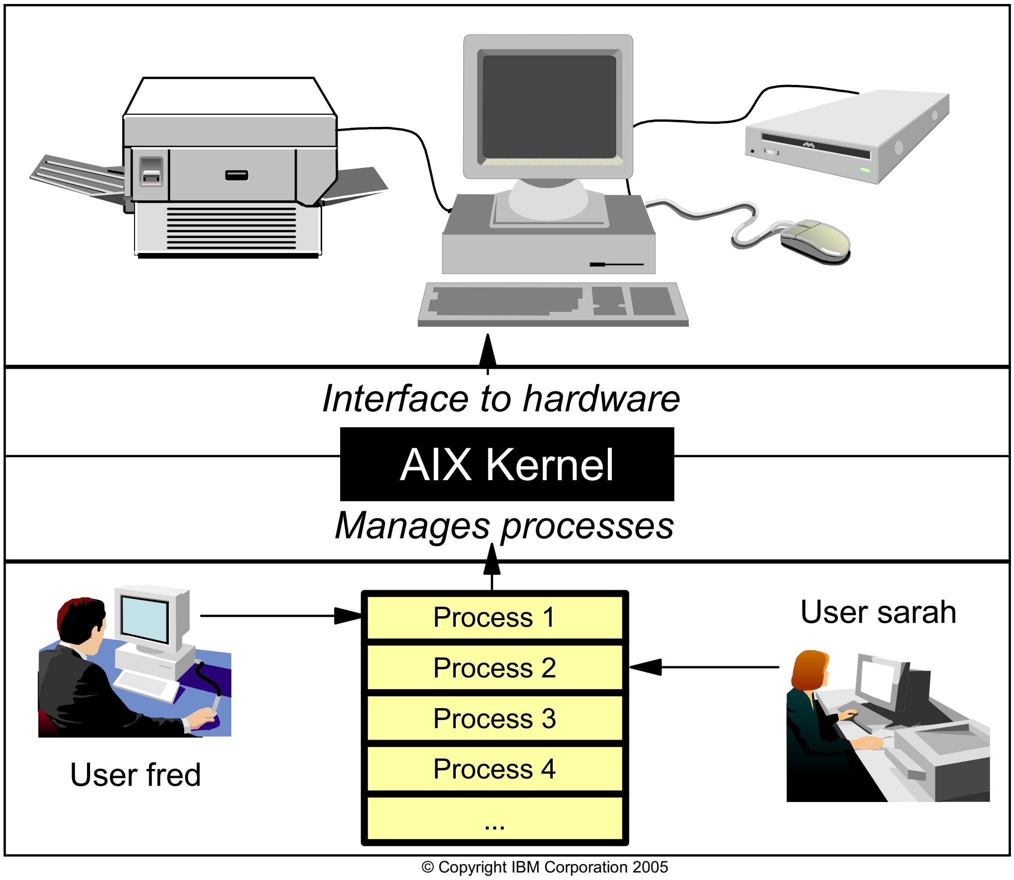


Figure 1-2. AIX Operating System AU139.0

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.

Working on an AIX System (1 of 2)

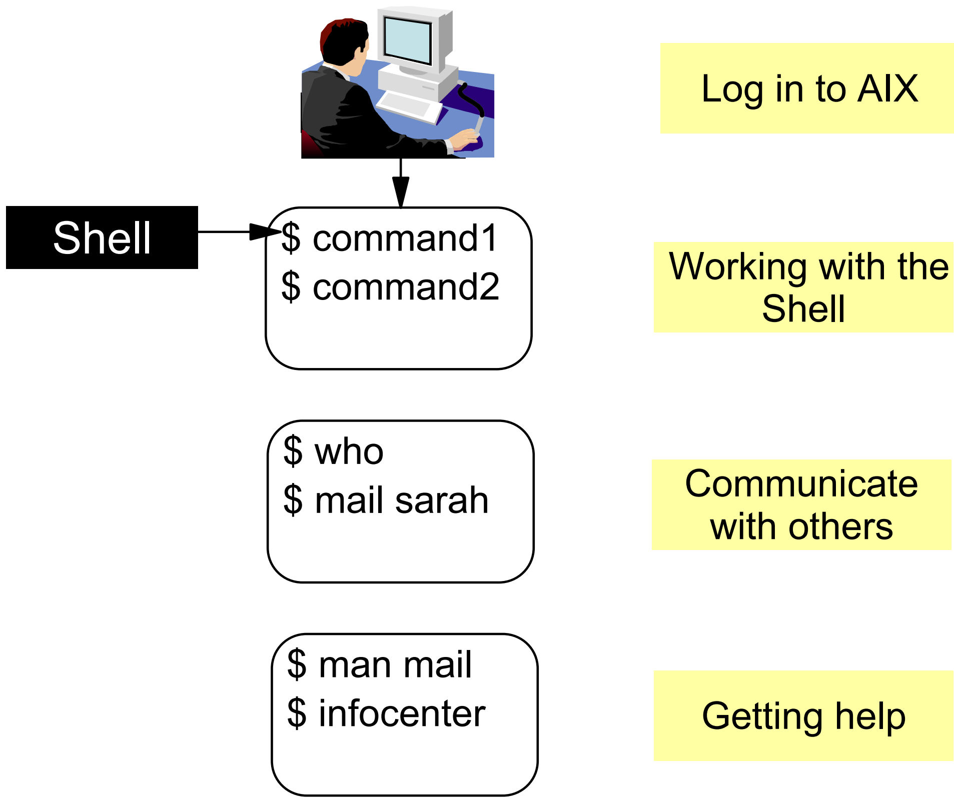


Figure 1-3. Working on an AIX System (1 of 2) AU139.0

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 his 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 On-line Documentation*. How to work with these help tools is also a major topic in this course.

Working on an AIX System (2 of 2)

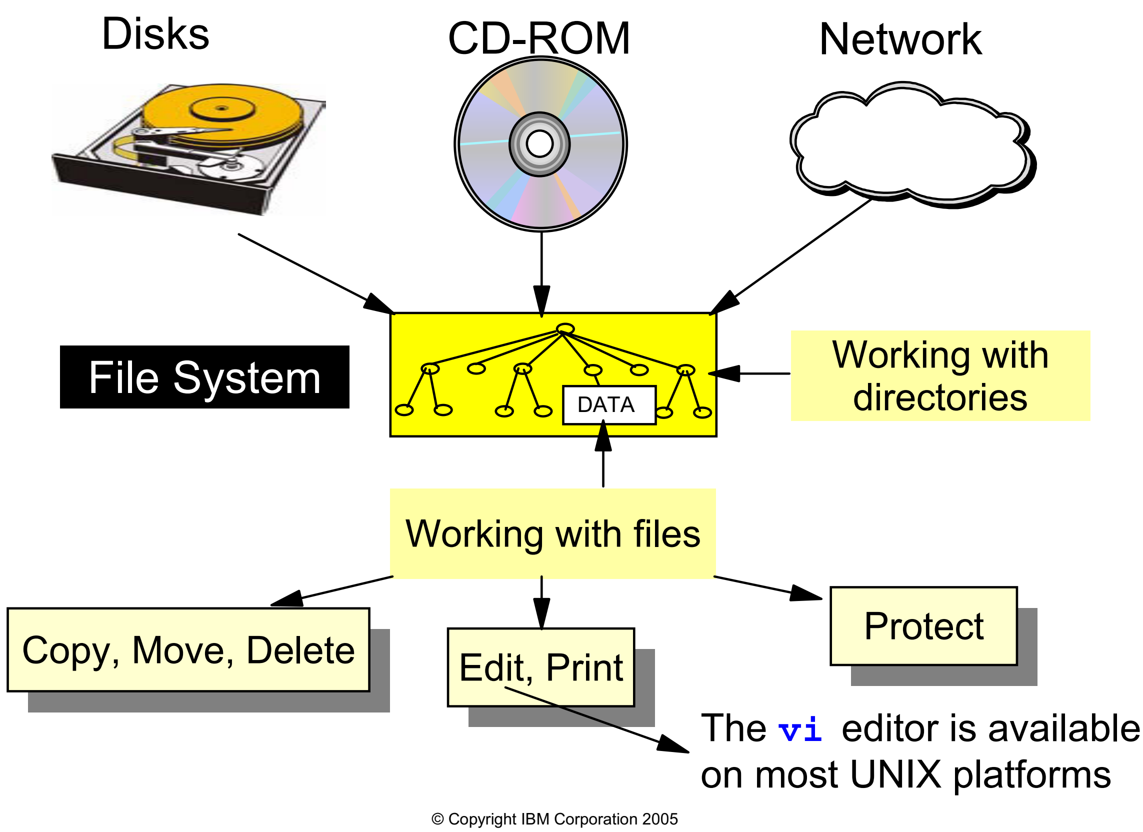


Figure 1-4. Working on an AIX System (2 of 2) AU139.0

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.

Files 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.

Activity: Fill in the Blanks

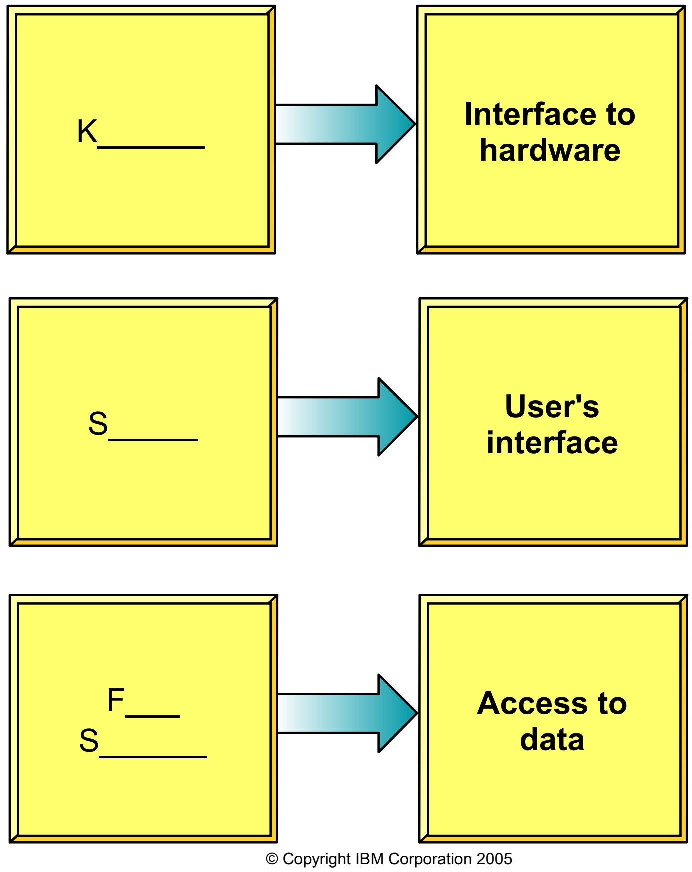


Figure 1-5. Activity: Fill in the Blanks AU139.0

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.

The Shell: User Interface to AIX

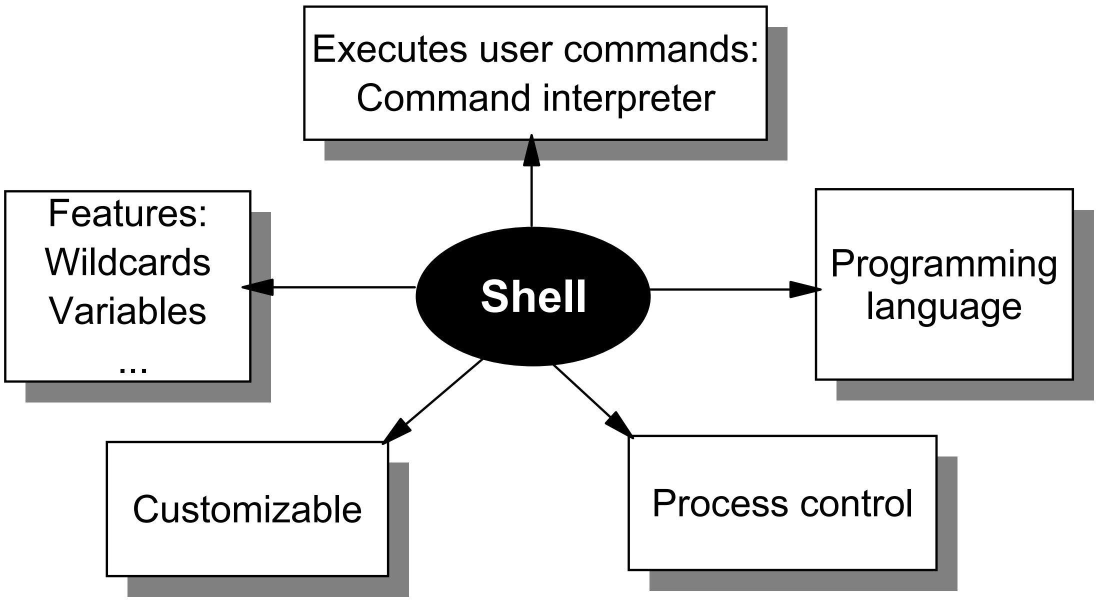


Figure 1-6. The Shell: User Interface to AIX AU139.0

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.

Useful AIX Utilities

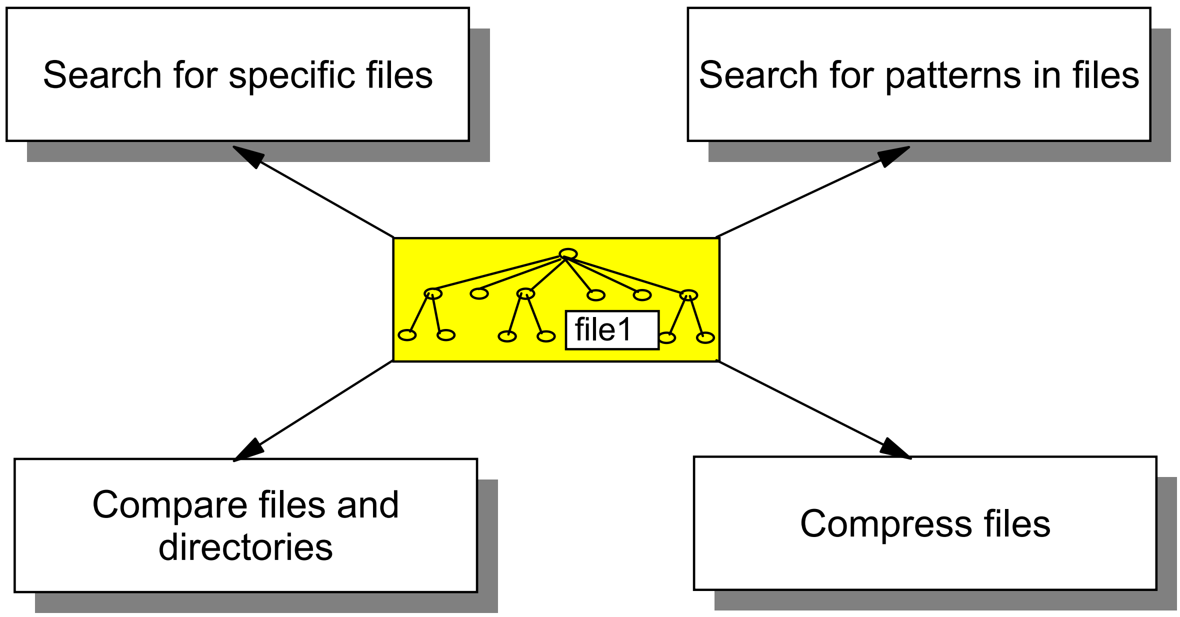


Figure 1-7. Useful AIX Utilities AU139.0

Notes:

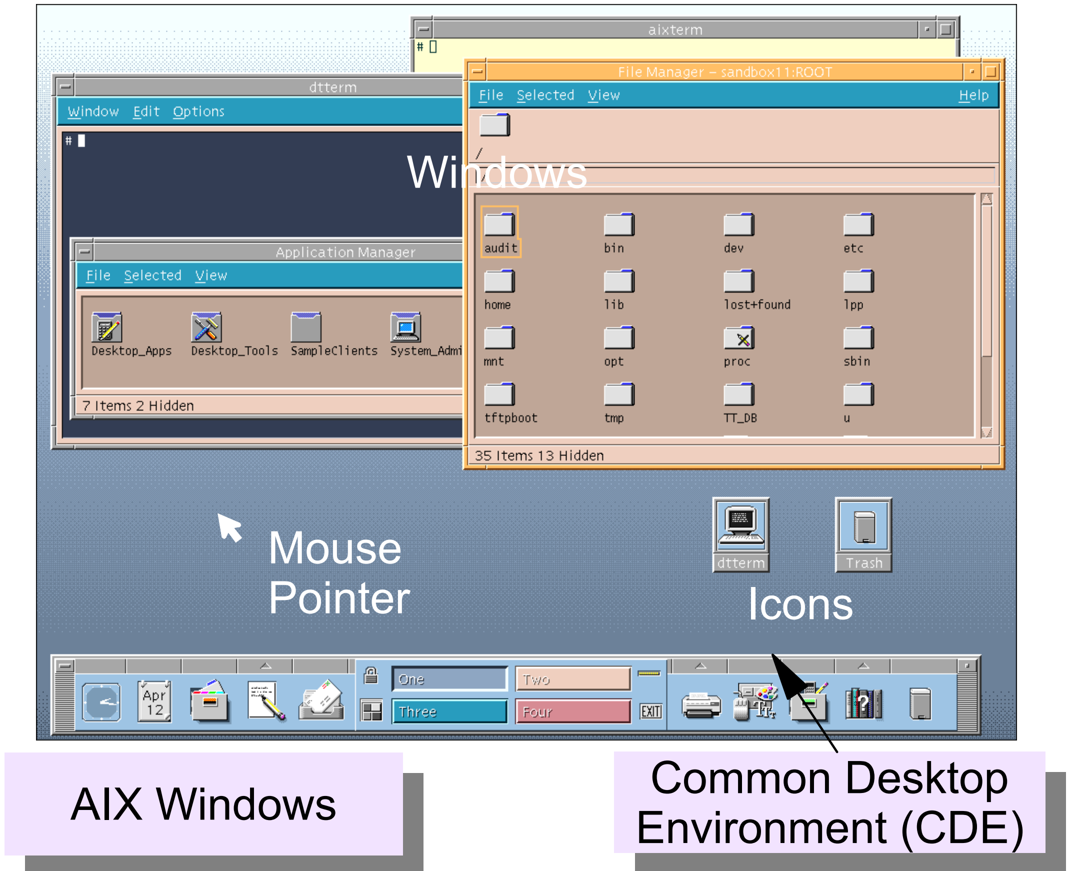
AIX utilities

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.

AIX Graphical User Interfaces



© Copyright IBM Corporation 2005

Figure 1-8. AIX Graphical User Interfaces AU139.0

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)

Using and customizing these desktops are major topics in 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

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:
   a.
   b.

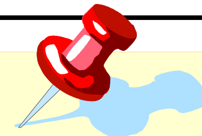
5. True or false: AIX only supports file systems on hard disks

Figure 1-9. Checkpoint AU139.0

Notes:

Take some time and try to answer the questions.

Unit Summary



Having completed this unit, you should be able to:

- 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 a 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.

Figure 1-10. Unit Summary AU139.0

Notes:

Unit 2. Using the System

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

Figure 2-1. Unit Objectives AU139.0

Notes:

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:
```


Figure 2-2. Logging In and Out AU139.0

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 into 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).

`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.

Passwords

- Creating or Changing:

```
$ passwd

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

$
```


Figure 2-3. Passwords AU139.0

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. 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.

Command Format

AIX commands have the following format:

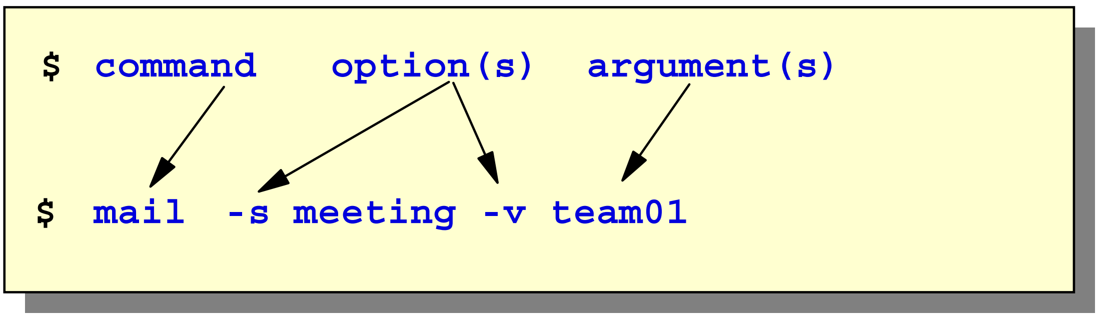


Figure 2-4. Command Format AU139.0

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:

`$ mail`	just command
`$ mail -f`	command and option
`$ mail team01`	command and argument

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!!

Figure 2-5. Command Format Examples AU139.0

Notes:

Introduction

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

The `date` and `cal` Commands

- **Checking the date:**

```
$ date
Tue  Jan 14  10:15:00GMT   2003
```

- **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 2003
```


Figure 2-6. The `date` and `cal` Commands AU139.0

Notes:

Introduction

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

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
```


Figure 2-7. The `clear`, `echo`, and `banner` Commands AU139.0

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

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 ?

Figure 2-8. Activity: Questions and Answers AU139.0

Notes:

Questions

Take some time and answer the questions.

The `who` and `finger` Commands

- Finding who is on the system:

```
$ who
root      lft0      Sept 4 14:29
team01pts/0    Sept 4 17:21
```

- Finding who you are:

```
$ who am i
team01   pts/0    Sept 4 17:21         (or)

$ 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.
```


Figure 2-9. The `who` and `finger` Commands AU139.0

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.

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>
```


Figure 2-10. Sending Mail AU139.0

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>

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 Tues 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)

Figure 2-11. Receiving Mail AU139.0

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`	appends messages to a file
`t`	display a message

There are many more. To obtain a the list of commands available type in a `?` at the `?` prompt or see the *AIX 5L Version 5.3: 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.

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.

The `write` and `wall` Commands

- Send **messages** to other **users** on a system

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

  `write` provides *conversation-like* communication with another logged-in user. Each user alternatively sends and receives messages.

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

  For example:

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

Figure 2-12. The `write` and `wall` Commands AU139.0

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 into 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>**

o **<enter>**

This starts the conversation. The o 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>**

oo **<enter>**

The oo 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>.**

`talk` With Another User

$ talk fred (user john requests the connection)

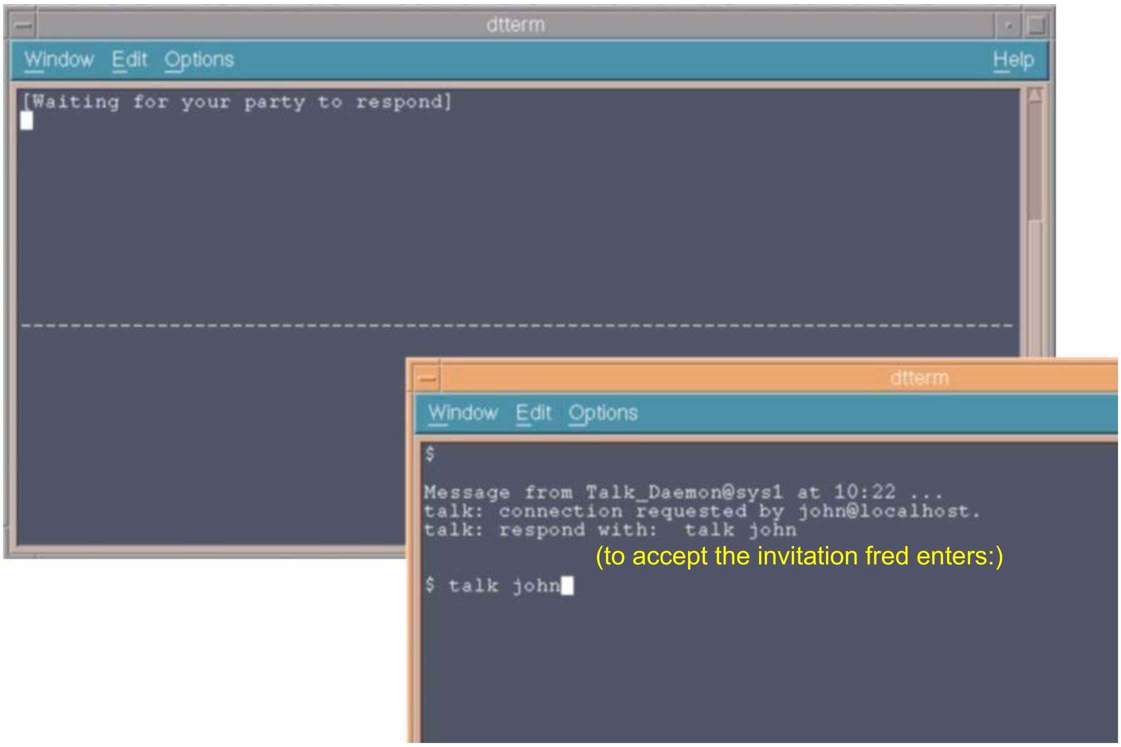


Figure 2-13. `talk` With Another User AU139.0

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`**.

`mesg`

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

`$ mesg n`	Refuses messages
`$ mesg y`	Permits messages

Figure 2-14. `mesg` AU139.0

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.

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

Figure 2-15. Keyboard Tips AU139.0

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.

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
   ```


Figure 2-16. Checkpoint AU139.0

Notes:

Exercise: Using The System

Figure 2-17. Exercise: Using The System AU139.0

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

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`**.

Figure 2-18. Unit Summary AU139.0

Notes:

Unit 3. AIX 5L V5.3 Documentation

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 5L V5.3 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 5L V5.3 Web-based online documentation

Figure 3-1. Unit Objectives AU139.0

Notes:

`man` Command

- The `man` command provides reference information on **commands**, **subroutines** and **files**

- Manual information consists of:

  - ***PURPOSE*** (one-line description)
  - ***SYNTAX*** (syntax)
  - ***DESCRIPTION***
  - ***FLAGS***
  - ***Examples*** (sample commands)
  - ***FILES*** (associated files)
  - ***RELATED INFORMATION***

Figure 3-2. `man` Command AU139.0

Notes:

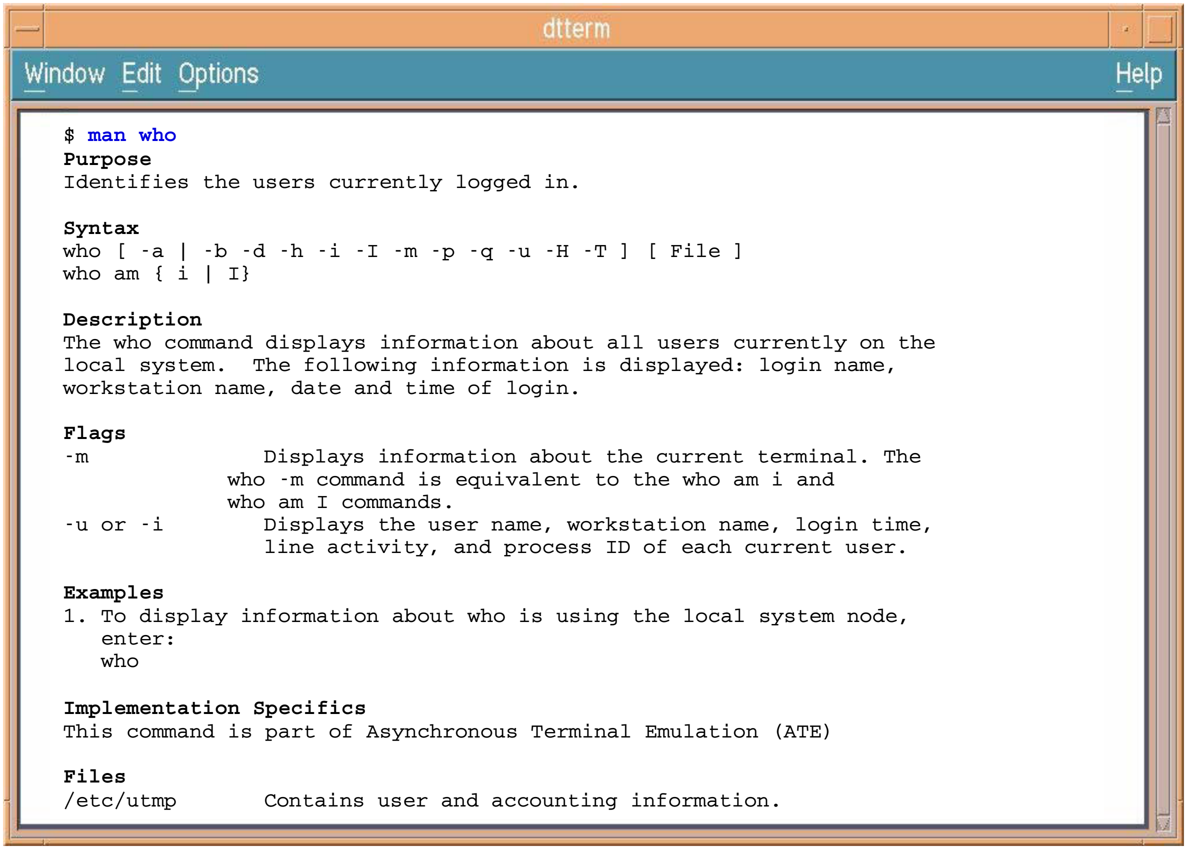
`man` command features

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	The syntax of the command
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

`man` Example




```
$ 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 AU139.0

Notes:

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

`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:

dtterm

Window Edit Options Help

```
$ 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
```


Figure 3-4. *man -k*: Working with a Keyword AU139.0

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`.

Viewing AIX 5L V5.3 Documentation

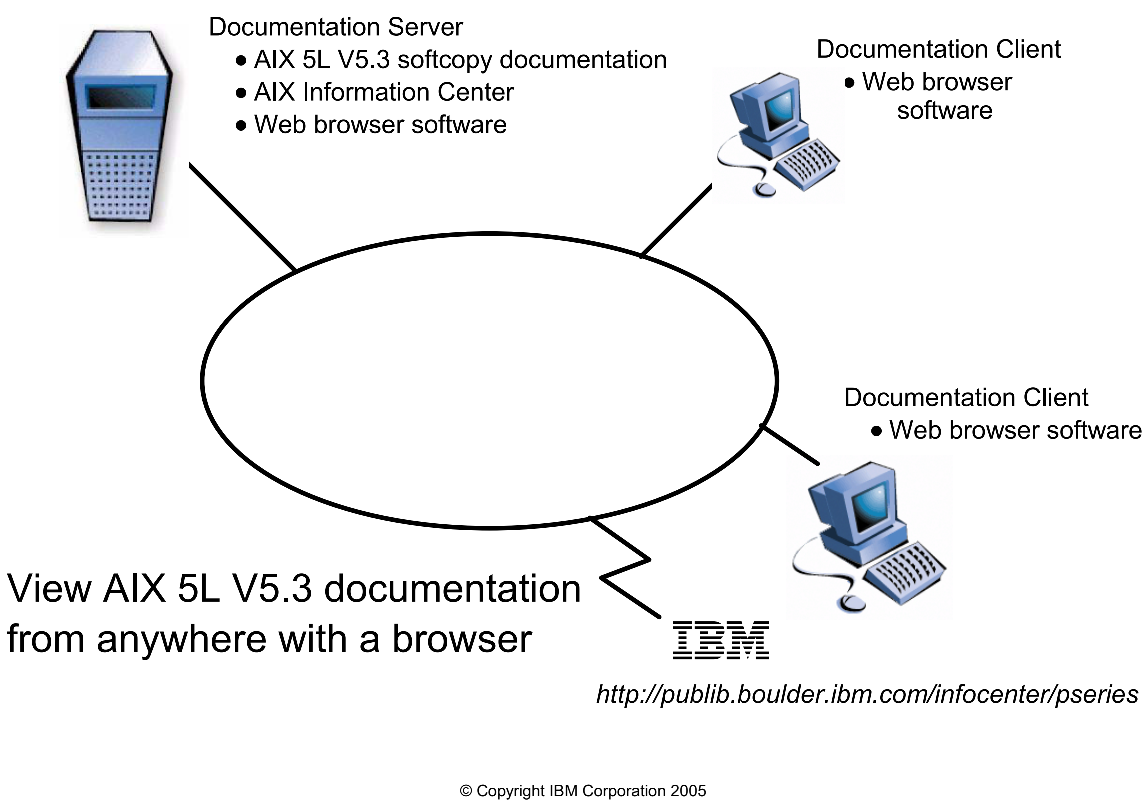


Figure 3-5. Viewing AIX 5L V5.3 Documentation AU139.0

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*. 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 (for example, the Mozilla browser) 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.

Accessing the Documents from a Web Browser

- Mozilla support added to AIX 5L V5.3
  - Current version can be downloaded from:
    - *http://www.ibm.com/servers/aix/browsers*
  - Support for Netscape dropped as of AIX 5L V5.3
- Mozilla is usually setup as the default browser for AIX 5L V5.3 documentation
- Additional Mozilla information at:
  - *http://www.ibm.com/servers/aix/browsers*

Figure 3-6. Accessing the Documents from a Web Browser AU139.0

Notes:

Web Browser

As of AIX 5L V5.3, the Mozilla Web browser is the default browser for AIX. It is not shipped with the system, but can be ordered on a separate CD. Or downloaded from a Web site.

As of release AIX 5L V5.3, Netscape is no longer supported.

Install Information Center

- Install documents
  - Installed as part of the base install
  - Installed afterwards with Configuration Assistant
- Information Center options
  - Standalone
  - Documentation server
  - Remote documentation server

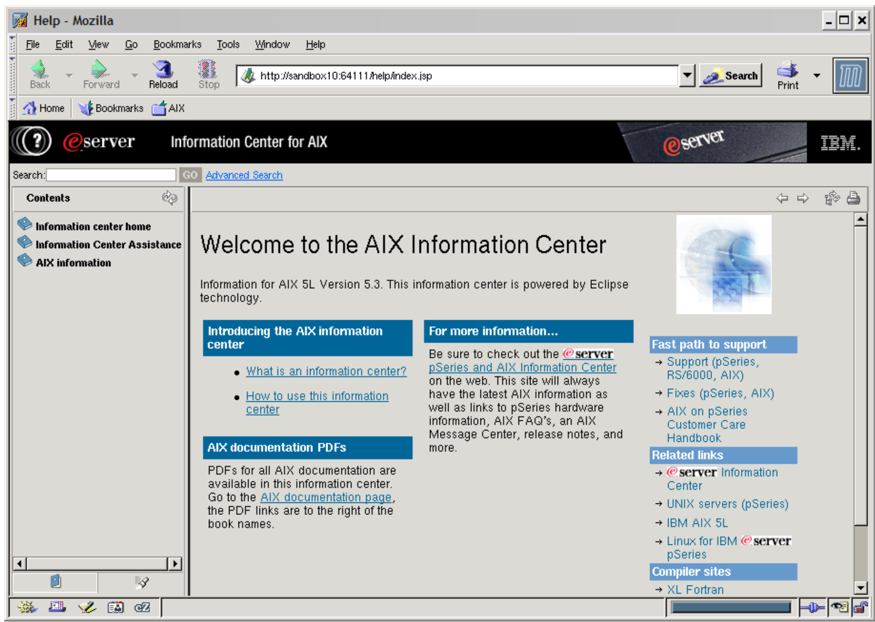


Figure 3-7. Install Information Center AU139.0

Notes:

AIX Information Center

The AIX Information Center is new with AIX 5L V5.3. It is accessed from the Web at *http://publib.boulder.ibm.com/infocenter/pseries*. The AIX 5L V5.3 documentation is a part of the information center. (Select the AIX documentation page)

The AIX Information Center can be installed on your system. They will have the same look and function as the Web site mentioned above only the actual documents will reside on the server you configure. There will be links that will resolve to external Web sites that may not be reachable if you are on a private network.

Information Center installation

The installation center can be installed as part of the OS, installed after BOS install with the Configuration Assistant or installed like any other software product using the standard installation tools.

AIX 5L V5.3 Documentation

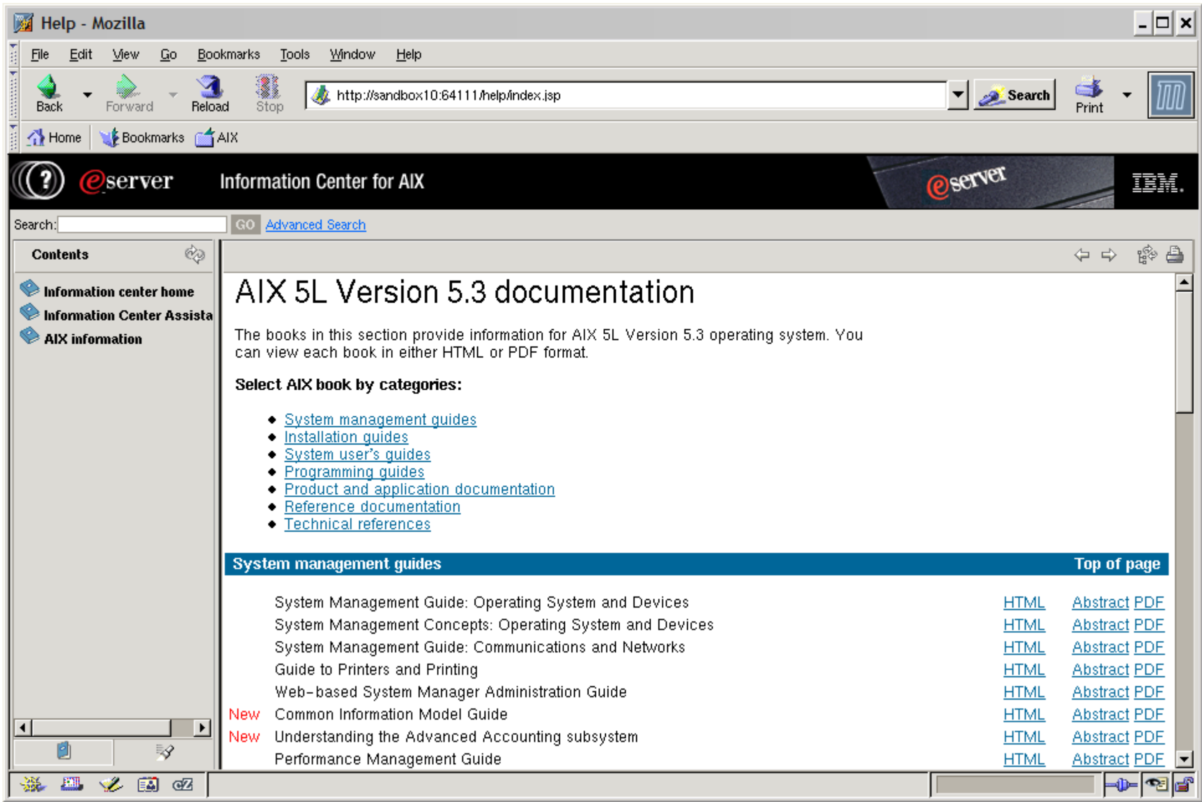


Figure 3-8. AIX 5L V5.3 Documentation AU139.0

Notes:

Accessing documentation

Once the documentation is set up, it can be accessed with the `infocenter` command. These examples show the documentation being accessed from a local system.

Web access

If the documentation was not installed on your system, online documentation is also available at:

http://publib.boulder.ibm.com/infocenter/pseries

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-hand 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 use the `Advanced Search` tool 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 browsers `Print` function usually found in the `File` menu.

Search AIX 5L V5.3 Documentation

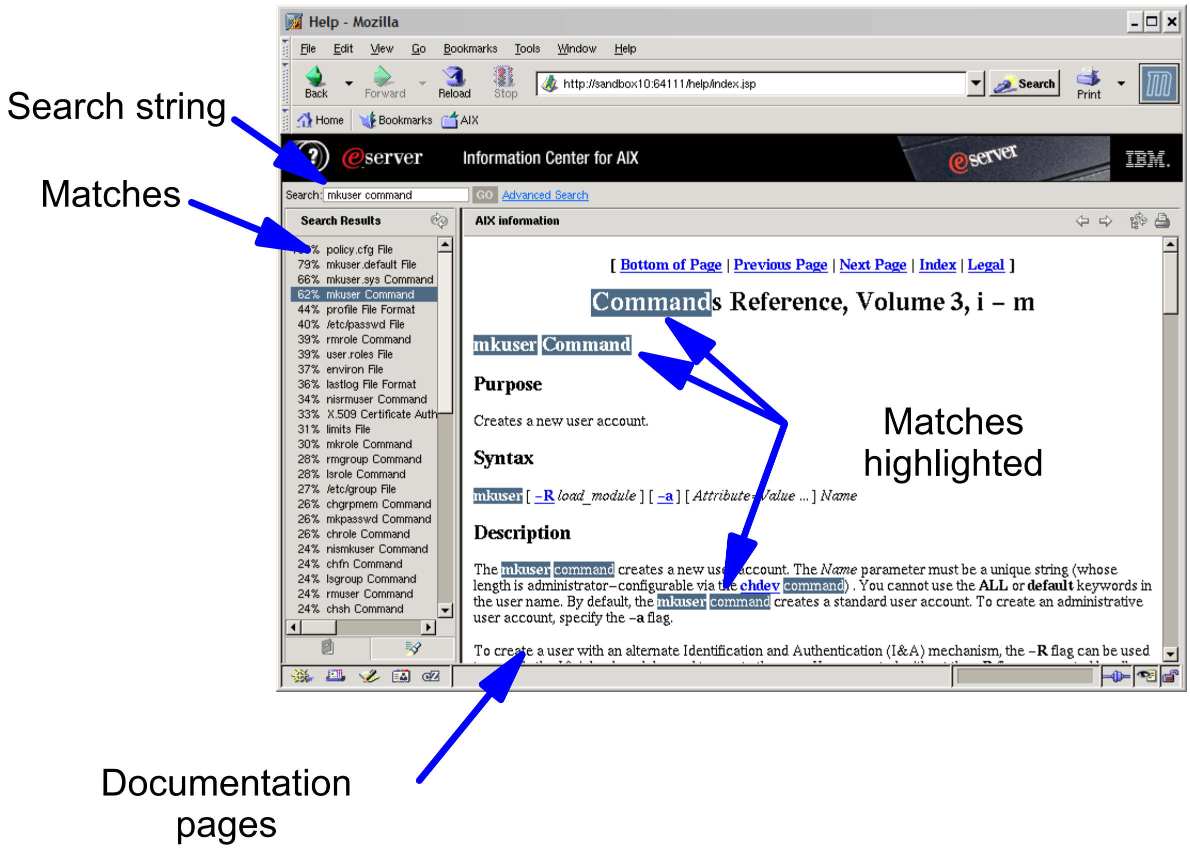


Figure 3-9. Search AIX 5L V5.3 Documentation AU139.0

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. Notice the list on the left hand side of the screen. These are the matches from the search in order of quality with the best match at the top of the list. Select the match that you wish and the text from the document will be displayed on the left with the search key words highlighted.

Note: The first time a search is done after the install, the indexes must be built. This may take some time to build depending on the type of system.

Advanced Search Options

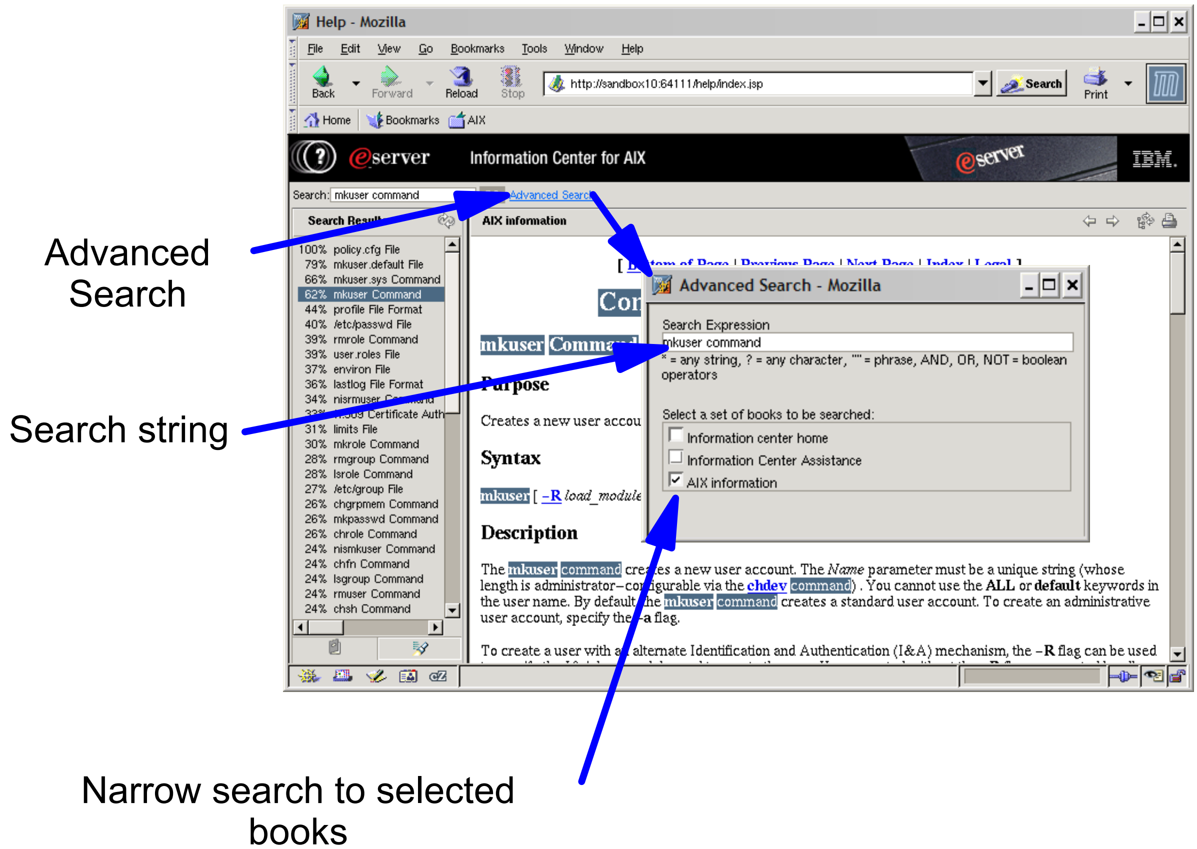


Figure 3-10. Advanced Search Options AU139.0

Notes:

Advanced search

By selecting the `Advanced Search` button, you can narrow the search to a subset of documents.

The Advanced Search panel also provides hints on how to build wildcard and combination search expressions.

Checkpoint

1. Which command displays manual entries online?

2. Complete the following sentences:
   The AIX 5L V5.3 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?

Figure 3-11. Checkpoint AU139.0

Notes:

Exercise: AIX 5L V5.3 Documentation

Figure 3-12. Exercise: AIX 5L V5.3 Documentation AU139.0

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

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 5L V5.3
- The on-line documentation and pSeries InfoCenter use the same interface
- Mozilla is the Web browser shipped with AIX 5L V5.3

Figure 3-13. Unit Summary AU139.0

Notes:

Unit 4. Files and Directories

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

Figure 4-1. Unit Objectives AU139.0

Notes:

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

Figure 4-2. A File AU139.0

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.

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: **CD-ROM-Device** is represented by **/dev/cd0**

Figure 4-3. File Types AU139.0

Notes:

Various file types

An *ordinary* file can contain either text or code data. Text files are readable by a user and can be displayed or printed. Code data, also known as a *binary file* is readable by the computer. Binary files may be executable.

Directories contain information the system needs to access all types of files, but they do not contain the actual data. Each directory entry represents either a file or a subdirectory.

Special files usually represent devices used by the system.

Directory Contents

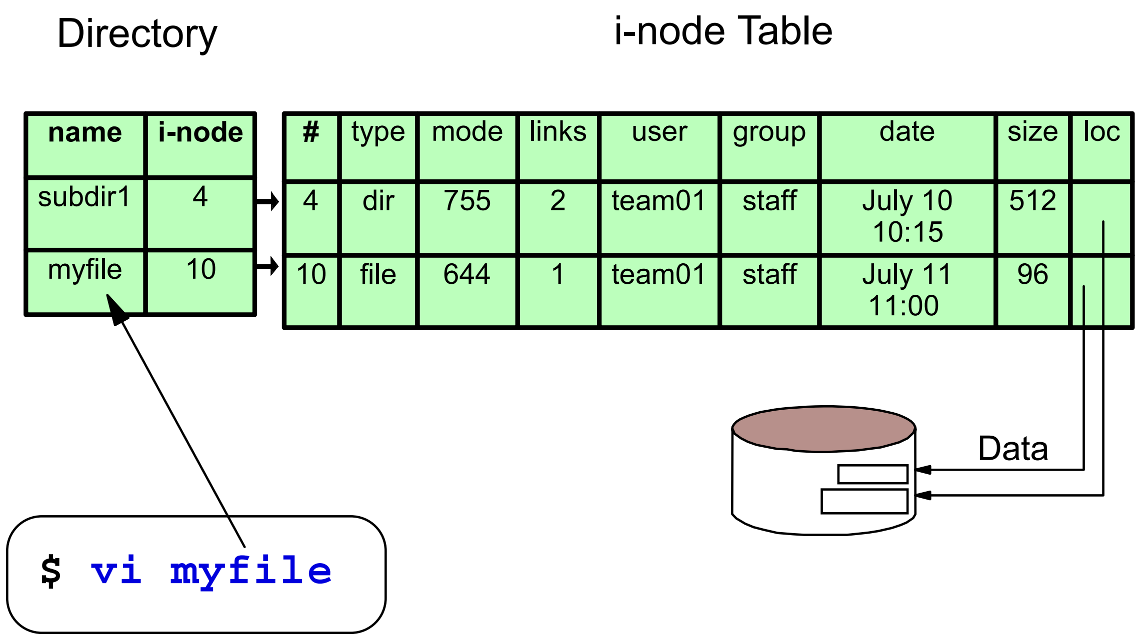


Figure 4-4. Directory Contents AU139.0

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.

Hierarchical Structure

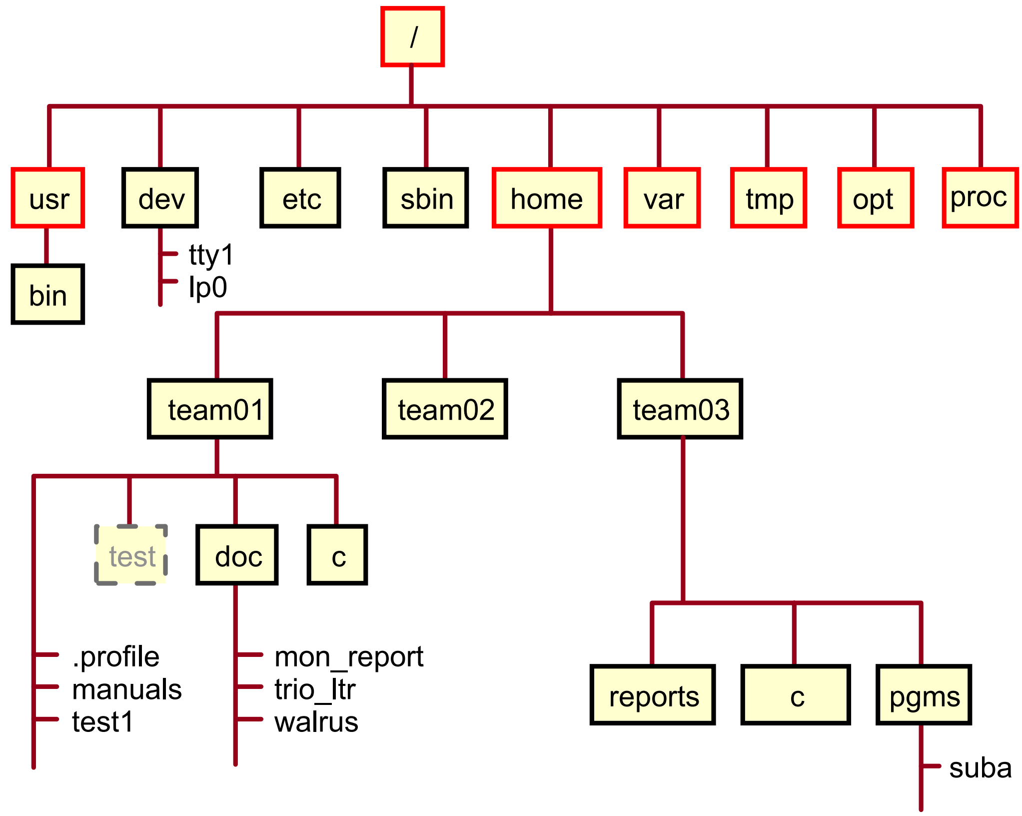


Figure 4-5. Hierarchical Structure AU139.0

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** - System utilities for system startup

/**dev** - Special files that represent devices

/**etc** - System configuration files used by system administrators

The /**usr** directory contains system programs such as:

/**usr**/**bin** - User commands such as `ls`, `cat`, `date`

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

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

The /**tmp** directory will hold files that are temporarily needed or created by applications and programs.

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.

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.

Path Names

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

- Full path name (start from the /-directory):

```
/home/team01/doc/mon_report
/usr/bin/ls
```

- Relative path name (start from current directory):

```
./test1 or test1          (. = current directory)
../team03/.profile        (.. = parent directory)
```


Figure 4-6. Path Names AU139.0

Notes:

File path names

The path name is written as a string of names separated by forward slashes (/)(not back slashes (\) like in DOS or OS/2). The right-most name can be any type of file. The other names must be directories.

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

Where Am I?

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

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


Figure 4-7. Where Am I? AU139.0

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).

Listing Directories

Syntax : **ls** [directory]

To list the contents of your current directory:

```
$ ls
c  doc  manuals  test1
```

To list all files, including hidden (.) files:

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

To list all files to the end of the directory tree:

```
$ ls -R
c   doc   manuals   test1

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


Figure 4-8. Listing Directories AU139.0

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 as well, the `-R` option can be used.

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
```


Figure 4-9. Long Listing of Files AU139.0

Notes:

File listing details

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 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 was 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.

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 ..`

Figure 4-10. Change Current Directory AU139.0

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.

Activity: Q + A

1. How can you determine the **inode number** of a file ?
2. Where are the **names** and **inode 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** ?

   ../team03/dir1 :
   /tmp/file1 :
   /.profile :
   ./.profile :
7. Write down the **three different file types** that AIX knows:
   a)
   b)
   c)

Figure 4-11. Activity: Q + A AU139.0

Notes:

Take some time and answer the questions.

Your instructor will review the questions with you afterwards.

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

Figure 4-12. Creating Directories AU139.0

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.

Removing Directories

Syntax : `rmdir directory`

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

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

The directory must be empty !

```
$ rmdir doc
rmdir: doc not empty
```


Figure 4-13. Removing Directories AU139.0

Notes:

Removing directories

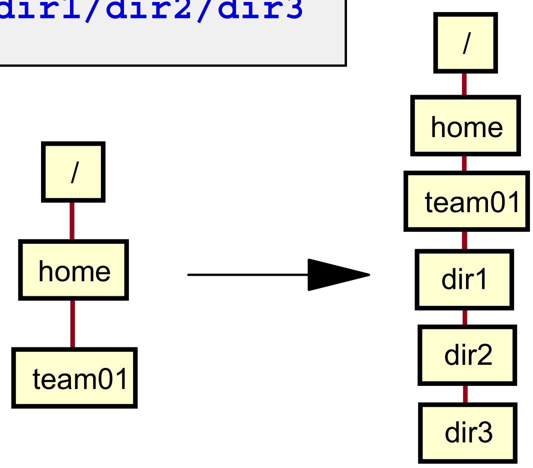
You get no message if the command is successful. It never hurts to follow a command such as this with an `ls`, which is discussed on the next page, to make sure that you have accomplished what you set out to do.

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

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
```


Figure 4-14. Working with Multiple Directories AU139.0

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.

Displaying Directory Information

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


$ istat 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
```


Figure 4-15. Displaying Directory Information AU139.0

Notes:

Using the `istat` 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: `ls -lc`
- To display the modification time: `ls -l`
- To display the access time: `ls -lu`

`istat` displays the i-node information for a particular file or directory. AIX systems maintain three timestamps for files and directories. The difference between an update and a modification is updated changes the i-node information; whereas, a modification changes 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.

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**

Figure 4-16. AIX File Names AU139.0

Notes:

AIX file names

Remember that 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.

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
```


Figure 4-17. touch Command AU139.0

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.

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?

Figure 4-18. Checkpoint (1 of 2) AU139.0

Notes:

Checkpoint (2 of 2)

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

-a	__ Provides a long listing of files
-i	__ Will list hidden files
-d	__ List subdirectories and their contents recursively
-l	__ Displays the inode 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

Figure 4-19. Checkpoint (2 of 2) AU139.0

Notes:

Exercise: Files and Directories

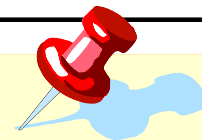
Figure 4-20. Exercise: Files and Directories AU139.0

Notes:

After this exercise, you will be able to:

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

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 full path name always begins with a / (forward slash).
- The following commands can be used with directories: **`pwd`**, **`cd`**, **`mkdir`**, **`rmdir`** and **`ls`**.

Figure 4-21. Unit Summary AU139.0

Notes:

Unit 5. Using Files

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** 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

Figure 5-1. Unit Objectives AU139.0

Notes:

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
```

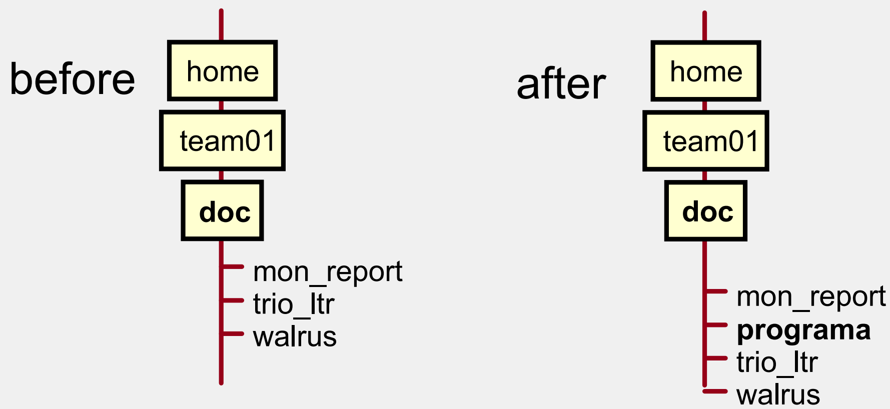


Figure 5-2. Copying Files AU139.0

Notes:

Copying files

This visuals introduces the `cp` command which is used to make a copy of a file. The left picture represents the file structure before the `cp` command. The right picture shows the file structure after executing the copy.

Examples (1 of 2)

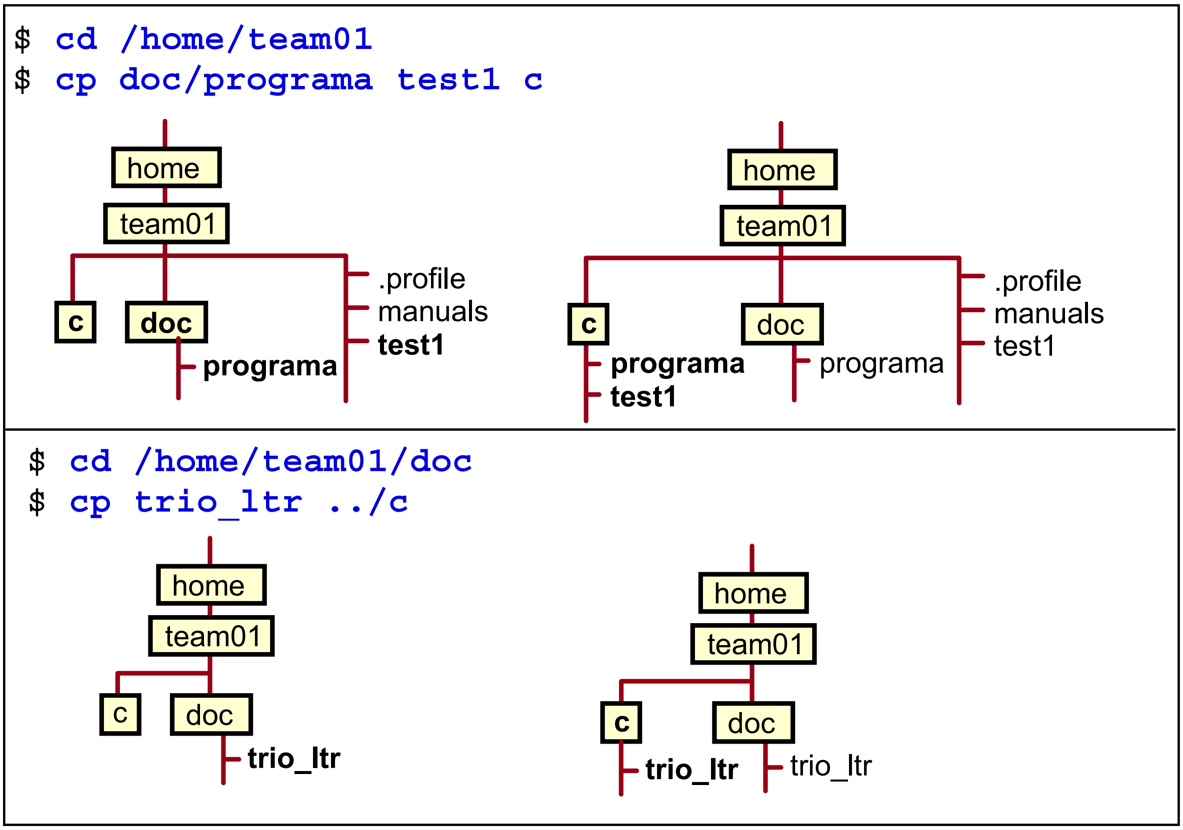


Figure 5-3. Examples (1 of 2) AU139.0

Notes:

Copying multiple files

If you are copying more than one file in one operation, then the specified 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 write over the original contents of the file without warning. To avoid this use `cp -i` (interactive copy).

Target is a directory

If the target is a directory, the copies of the files will be placed into that directory and will have the same file names as the original.

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.)

Moving and Renaming Files

```
mv source target
mv file1 file2 ... target_dir
```

```
$ pwd
/home/team01/c
$ mv trio_ltr t.letter
```

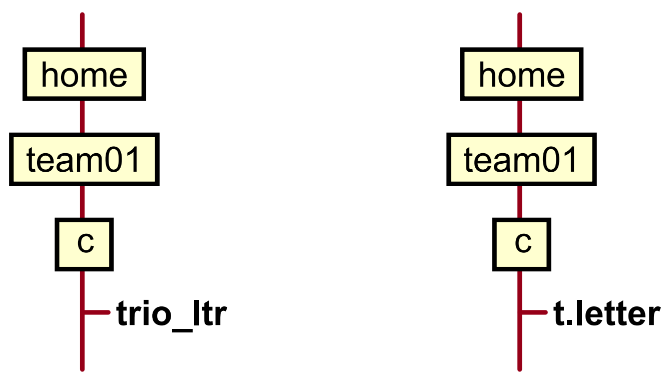


Figure 5-4. Moving and Renaming Files AU139.0

Notes:

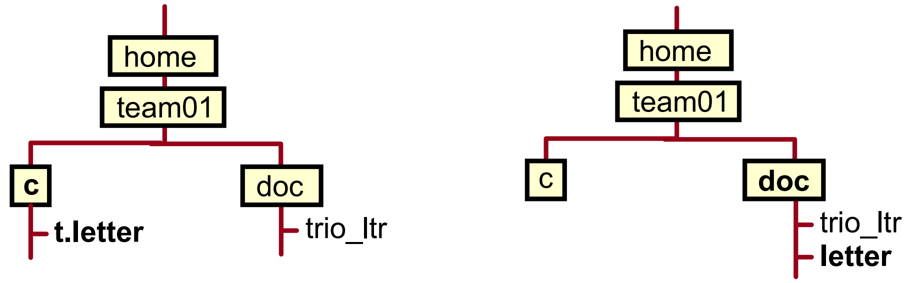
The mv command

The move command (mv) moves files from one directory to another or to change the name used under the current directory.

There is no rename command available in AIX. Renaming is done with the mv command.

Examples (2 of 2)

```
$ pwd
/home/team01/c
$ mv t.letter ../doc/letter
```




```
$ pwd
/home/team01/c
$ mv ../doc/mon_report .
```

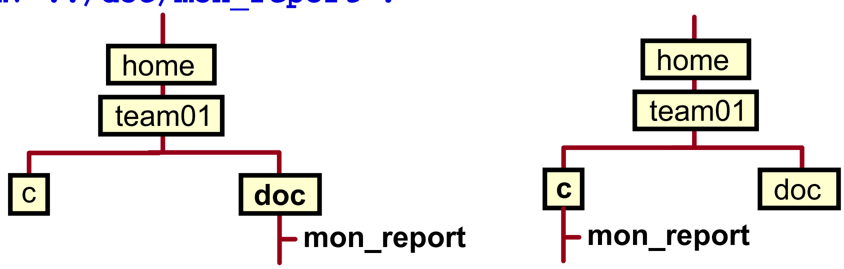


Figure 5-5. Examples (2 of 2) AU139.0

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/or 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.

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)
```


Figure 5-6. Listing File Contents AU139.0

Notes:

Introduction

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

Too much output?

The problem with this command is that it does not paginate the output, but displays it all at once. If the output from the `cat` command 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 with the `cat` command.

Displaying Files

```
pg filename
more filename
```

```
$ pg 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."
:   <Enter>
```

```
$ more 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."
walrus (100%)   <Enter>
```

one page at a time

Figure 5-7. Displaying Files AU139.0

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.

When **more** is reading from a file, it displays a `%` with the `More` prompt. This provides the fraction of the file (in characters) that the **more** command has read. Pressing **h** will display help information.

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

Figure 5-8. wc Command AU139.0

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.

Activity: Working with the `wc` Command

Figure 5-9. Activity: Working with the wc Command AU139.0

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 testfile1`**

Linking Files

```
ln source_file target_file
```

The **ln** command allows one file to have **more than one name**:

```
$ pwd
/home/team01

$ ln manuals /home/team02/man_files
```

Both copies use the same i-node

Figure 5-10. Linking Files AU139.0

Notes:

Introduction

The `ln` command in its simplest of forms allows one file to have two or more different names in the tree structure; that is, an alternate name.

It should be noted that the owner of the file remains the same as do the permissions.

When using the `ln` command, always provide the currently existing filename as the `source_file` and provide the new filename that is to be created as the `target_file`.

Removing Files

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

```
$ ls
mon_report   trio_ltr   walrus
$ rm mon_report
$ ls
trio_ltr   walrus
```

-i: Remove a file interactively

```
$ rm -i walrus
rm: Remove walrus: y
$ ls
trio_ltr
```


Figure 5-11. Removing Files AU139.0

Notes:

Methods of removing entries

The `rm` command removes the entries for the specified file or files from a directory. Note that the `rm` command may require confirmation from the user. For the interactive version of the command use the `-i` option.

The `-r` option permits recursive removal of directories and their contents if a directory is specified. Be careful when using this option as it does not require the directory to be empty in order for this option to work.

Printing Files

- To queue files to the printer use the **qprt** command:

```
$ qprt filename filename2 filename3 ....
```

- The **qchk** command displays the current status of a print queue:

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

- To cancel your print job use the **qcan** command:

```
$ qcan -x 99
```

Job number

Figure 5-12. Printing Files AU139.0

Notes:

Print subsystems

AIX 5L V5.1, introduced support for other printing subsystems by adding the System V Printing Subsystem. Previous versions of the operating system only support the AIX Printing System.

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
```

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.

Figure 5-13. Checkpoint AU139.0

Notes:

Exercise: Using Files

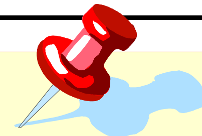
Figure 5-14. Exercise: Using Files AU139.0

Notes:

After completing the lab, you are able to:

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

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

Figure 5-15. Unit Summary AU139.0

Notes:

Unit 6. File Permissions

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

Figure 6-1. Unit Objectives AU139.0

Notes:

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

Figure 6-2. Long Listing of Files AU139.0

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 user name of person who owns entry
- Field 4 shows 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.

File Protection/Permissions

rwx	rwx	rwx
user	group	others

r = read   w = write   x = execute

- 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 use the file name as a command (r is also needed)
```

- For a **directory**:

```
r => Can find out what files are in the directory
w => Can create/remove files in the directory (x is needed)
x => Has permission to be in the directory
     (that is, cd to the directory or access files from the
      directory)
```


Figure 6-3. File Protection/Permissions AU139.0

Notes:

File permissions

- If the permissions have an **r**, you can look into the file.
- If you have a **w**, you can change the contents.
- If you have a script with an **x**., you also need the **r** permission.
- If the file contains executable code you need the **x** for execution, but the **r** permission is not necessary.
- For directories, **x** permission is required to access any of the files or subdirectories within it. This implies that **x** permission is required on all directories above it as well.
- For directories, **x** permission is NECESSARY for **w** permission to be effective. If you cannot get into the directory, then you cannot create or remove files within it.

Note: In order to remove a file you only need **x** and **w** permissions in the directory; you do not need *any* permissions on the file.

Changing Permissions (Symbolic Notation)

```
chmod  mode  filename
```

u = owner of the file **g** = owner's group **o** = other users on the system **a** = all	**+** : add permissions **-** : remove permissions **=** : clears permissions and sets to mode specified

```
$ ls -l newfile
-rw-r--r--   1  team01staff 58 Apr 2116:06 newfile

$ chmod go+w newfile
$ ls -l newfile
-rw-rw-rw-   1  team01staff 58 Apr 2116:06 newfile

$ chmod a+x newfile
$ ls -l newfile
-rwxrwxrwx   1  team01staff 58 Apr 2116:06 newfile

$ chmod o-rwx newfile
$ ls -l newfile
-rwxrwx---   1  team01staff 58 Apr 2116:06 newfile
```


Figure 6-4. Changing Permissions (Symbolic Notation) AU139.0

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.

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
```


Figure 6-5. Changing Permissions (Octal Notation) AU139.0

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.

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.

Default File Permissions

The default protections for newly created files and directories are:

File	**-rw-r--r--**	**644**
Directory	**drwxr-xr-x**	**755**

These default settings may be changed by changing the `umask` value.

Figure 6-6. Default File Permissions AU139.0

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.

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.

Figure 6-7. `umask` AU139.0

Notes:

Understanding permissions

`umask` is a command which set 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).

On a file, the execute permissions are never set.

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).

Activity: Personal Directories

Figure 6-8. Activity: Personal Directories AU139.0

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?

Write Permission on a Directory

```
$ ls -ld /home/team01
drwxrwxrwx 2 team01 staff  512  July 29  9:40  team01
$ ls -l /home/team01/file1
-rw-r--r-- 1 team01 staff 1300  July 30  10:30 file1
```

```
$ whoami
team02

$ vi /home/team01/file1
file1: The file has read permission only

$ vi myfile1
Ha! Ha! I changed this file. Figure out how.

$ mv myfile1 /home/team01/file1
override protection 644 for file1? y

$ cat /home/team01/file1
Ha! Ha! I changed this file. Figure out how.
```


Figure 6-9. Write Permission on a Directory AU139.0

Notes:

Write access to a directory

If you don't have write access to a file, you cannot change it; however, if you have write access to the directory in which this file resides, you can get around this.

team02 is able to change the contents of /**home**/**team01**/**file1** by moving and renaming another file into **team01**'s home directory. **team02** can execute this `mv` command because of write permission on **team01**'s /**home**/**team01** directory.

Allowing write access to a directory can be dangerous. If this is a security issue with your files, fix the gotcha by setting your `umask` correctly, using `chmod` to fix permissions of existing directories.

Function/Permissions Required

Command	Source Directory	Source File	Target Directory
`cd`	x	N/A	N/A
`ls`	r	N/A	N/A
`ls -l`	r, x	N/A	N/A
`mkdir`	x w (parent)	N/A	N/A
`rmdir`	x w (parent)	N/A	N/A
`cat, pg, more`	x	r	N/A
`mv`	x, w	NONE	x, w
`cp`	x	r	x, w
`touch`	x, w *	NONE	N/A
`rm`	x, w	NONE	N/A

Figure 6-10. Function/Permissions Required AU139.0

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.

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.
4. 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
```

5. Can Fred, who is a member of the finance group, modify the file **joblog**?

Figure 6-11. Checkpoint (1 of 3) AU139.0

Notes:

Checkpoint (2 of 3)

6. 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
```

Can Fred, who is a member of the finance group, modify the file **joblog**?

Figure 6-12. Checkpoint (2 of 3) AU139.0

Notes:

Checkpoint (3 of 3)

7. 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
```

Can Fred, who is a member of the finance group, copy the file **joblog** to his home directory?

Figure 6-13. Checkpoint (3 of 3) AU139.0

Notes:

Exercise: File Permissions

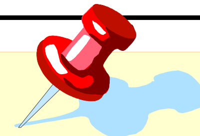
Figure 6-14. Exercise: File Permissions AU139.0

Notes:

After completing the exercise, you will be able to:

- Manipulate permissions on ordinary files and directories
- Interpret file and directory permission bits

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

Figure 6-15. Unit Summary AU139.0

Notes:

Unit 7. The vi Editor

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`

Figure 7-1. Unit Objectives AU139.0

Notes:

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**

Figure 7-2. Introduction to the `vi` Editor AU139.0

Notes:

The `vi` Editor

It is important to know `vi` for the following reasons:

- It is the only editor available in maintenance mode on RISC System/6000
- Standard editor across all UNIX systems
- Command-line editing feature
- Used as default editor for some programs

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`.

Starting `vi`

$ vi vifile

```
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"vifile"
[New file]
```

- If the file **"vifile"** does not exist, it will be created
- Otherwise, **vi** will open the existing file

Figure 7-3. Starting `vi` AU139.0

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 /**tmp** by default.
- If the file does not exist, an empty buffer is opened for this session.

The tildes characters represent empty lines in the editor.

The editor starts in command mode.

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.
~
~
~
~
~
~
~
~
~
~
~
```


Figure 7-4. Adding Text AU139.0

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

Once in text mode, any characters entered will be placed into the file.

To exit from text mode, press the `<Esc>` key.

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.
~
~
~
~
~
~
~
~
~
~
~
~
~
:wq
```

- To quit **without** saving: **:q!**
- To **save** and **exit**: **:x or :wq or <shift-zz>**

Figure 7-5. Exiting the Editor AU139.0

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

Cursor Movement

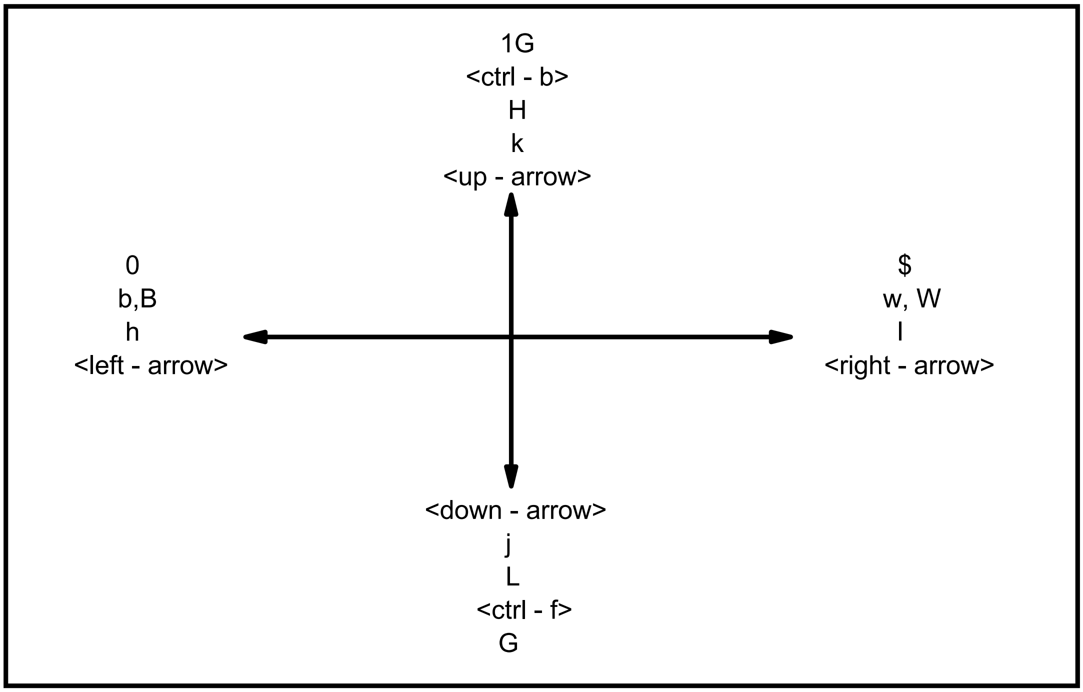


Figure 7-6. Cursor Movement AU139.0

Notes:

Moving within a file

Note that for all the upper case 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

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`

Figure 7-7. Deleting Text AU139.0

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.

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**

Figure 7-8. Search for a Pattern AU139.0

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.

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:	

Figure 7-9. Activity: `vi` Commands AU139.0

Notes:

Complete the table in the visual.

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`**
- Words can be changed: **`c2w`**
- Or every occurrence of a word can be substituted for another word(s)

Figure 7-10. Changing Text AU139.0

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 test mode and allow you to overtype the existing text beginning at the current cursor position. The `r` 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 place 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 of 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.

Moving Text

Commands

```
  This is the first line of text
_ This is the second line of text          dd
  This is the third line of text
```

```
  This is the first line of text
_ This is the third line of text
```

```
  This is the first line of text
  This is the third line of text           p
_ This is the second line of text
```

- **Moving** is done by **deleting the original text** using `dd`
- **Copying** is done by **yanking** text into a buffer
- In both cases, the text is pasted into its new location

Figure 7-11. Moving Text AU139.0

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.

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]
```

```
:r snacks
```

```
The following should be stocked in the
employee break room:

candy bars
soda pop
popcorn
```


Figure 7-12. **vi** - Executing AIX Commands AU139.0

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 don't want to have to rekey all the information or redirect the information after exiting the **vi** session, plus you can't 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`**.

`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

Figure 7-13. `vi` Options AU139.0

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 ai`	Sets autoindent on.
`:set noai`	Turns autoindent mode off.
`:set nu`	Enables line numbers.
`:set nonu`	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 ts=4`	Sets tabs to 4-character jumps.
`:set ic`	Ignores case sensitivity.
`:set noic`	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 (:)

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:

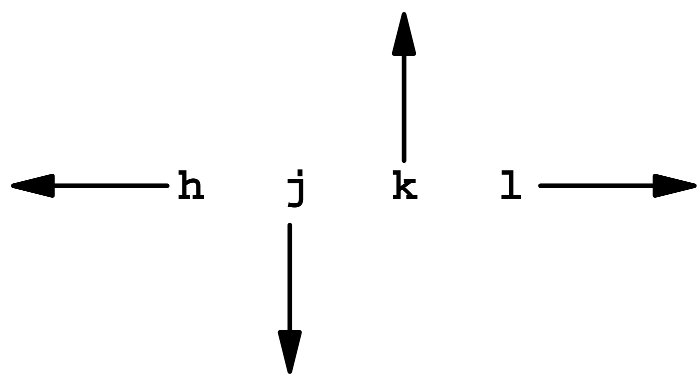


Figure 7-14. Command-Line Editing AU139.0

Notes:

Editing the command line

Normally, you can only edit the command line using the backspace or kill keys. 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.

If using the `emacs` editor, `set -o emacs` provides the same feature.

Editing commands

You can then edit the line as you would any line of text in a **vi** editing session.

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.

`vi` Editors

`vi`	Full-screen, full-function editor
`view`	Read only form of **`vi`**, changes cannot be saved unless overridden with a force (!)
`vedit`	Beginners 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`**

Figure 7-15. `vi` Editors AU139.0

Notes:

Introduction

`emacs` is another popular UNIX editor but is not standard across all UNIX platforms.

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?

Figure 7-16. Checkpoint AU139.0

Notes:

Exercise: `vi` Editor

Figure 7-17. Exercise: `vi` Editor AU139.0

Notes:

After completing the lab, you should be able to:

- Create and edit files using the `vi` editor
- Invoke *command line editing*

Unit Summary

Having completed this unit, you should be able to:

- 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.

Figure 7-18. Unit Summary AU139.0

Notes:

Unit 8. Shell Basics

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

Figure 8-1. Unit Objectives AU139.0

Notes:

The Shell

- **Korn (ksh)** or Bourne (bsh) or C (csh)
- **User interface** to AIX
- **Command interpreter**
- Enables **multiple tasks**
- Comprehensive **programming language**

Figure 8-2. The Shell AU139.0

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.

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:

  ```
  *   ?   !   [   ]   [   -   ]
  ```


Figure 8-3. Metacharacters and Wildcards AU139.0

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 `-l` then issuing the command `ls -l` would give you a long listing of the current directory because the `ls` command would think the `l` was an option rather than `-l` being a file name argument. Some AIX commands provide facilities to overcome this problem.

Available metacharacters

```
!  "  $  %  ^  &  *  (  )  {  }  [  ]  #  ~ ;  '  <  >  /  ?  '  \  |
```

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
```

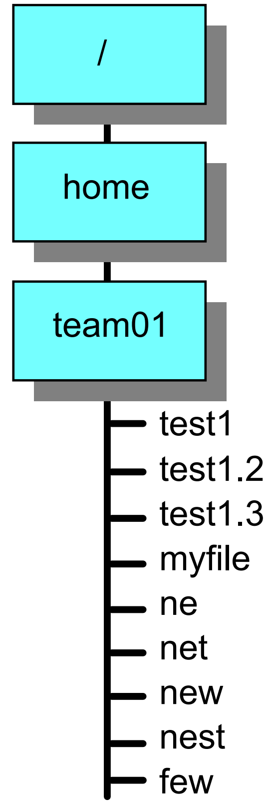


Figure 8-4. File Name Substitution (1 of 2) AU139.0

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.

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
```

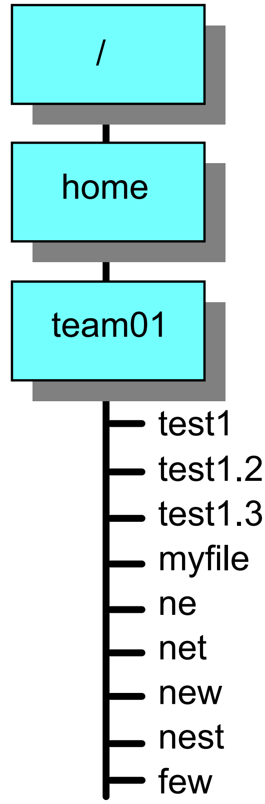


Figure 8-5. File Name Substitution (2 of 2) AU139.0

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.

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`**

The Standard Files

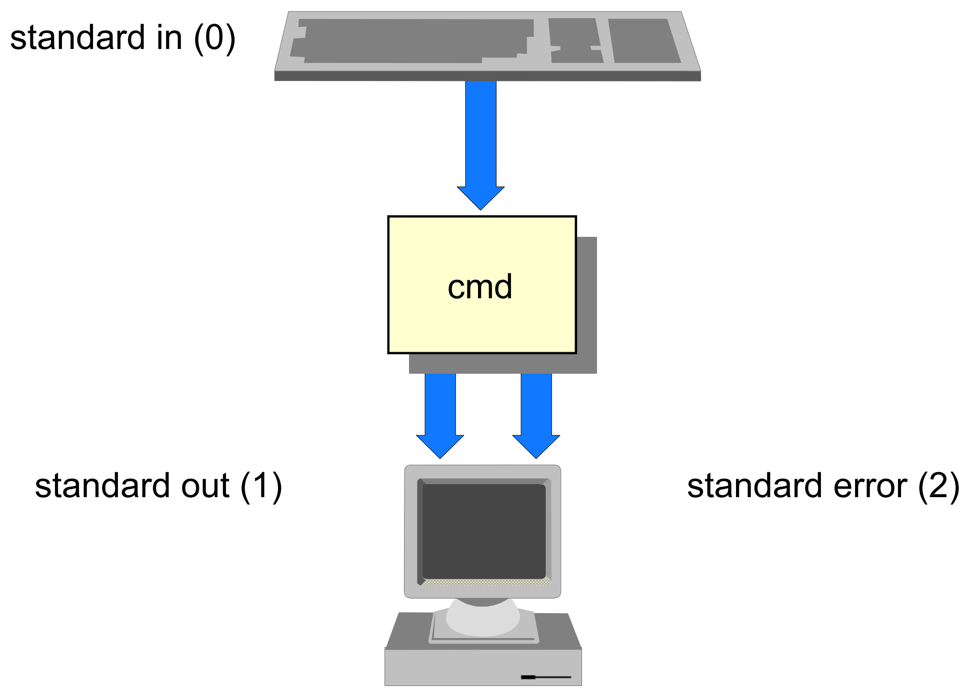


Figure 8-6. The Standard Files AU139.0

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`) is where the command expects to put its output, usually the screen. These defaults can be changed using redirection.

File Descriptors

- **Three descriptors** are assigned by the shell when the program starts:

Standard in:	<	0
Standard out:	>	1
Standard error:	2>	2

Figure 8-7. File Descriptors AU139.0

Notes:

AIX file descriptors

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**.

Input Redirection

- **Default standard input**

```
$ mail team01
Subject: Letter
This is a letter.
<ctrl-d>
Cc:
$
```

- **Redirect input from a file: <**

```
$ mail team01 < letter
$
```


Figure 8-8. Input Redirection AU139.0

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 subject -c Address(es) Address
```

The symbol < tells `mail` to take input from the file instead of the keyboard.

The `mail` program handles standard out differently than other commands.

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
$
```


Figure 8-9. Output Redirection AU139.0

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

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
```


Figure 8-10. Creating a File with `cat` AU139.0

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

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`

Figure 8-11. Activity: Review Shell Basics AU139.0

Notes:

Activity

Please answer the questions in the visual.

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
```


Figure 8-12. Error Redirection AU139.0

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.

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
```


Figure 8-13. Combined Redirection AU139.0

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.

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
```


Figure 8-14. Pipes AU139.0

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.

Filters

A **filter** is a command that **reads from standard in**, **transforms the input** in some way, and **writes to standard out**

Example:

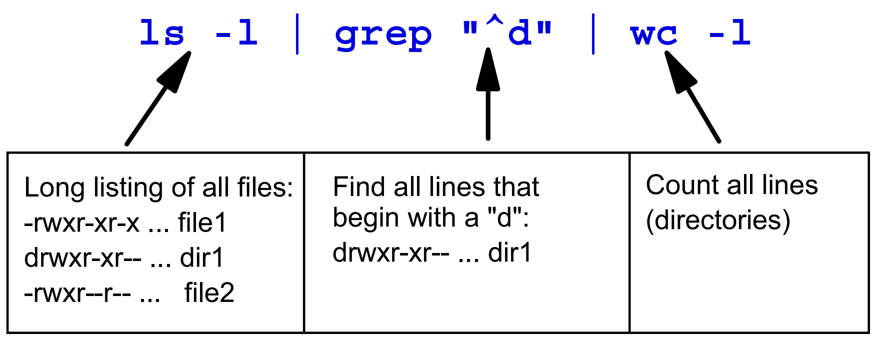


Figure 8-15. Filters AU139.0

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.

Split Outputs

The `tee` command reads standard input and sends the data to both standard output and a file

`ls | tee /tmp/ls.save | wc -l`

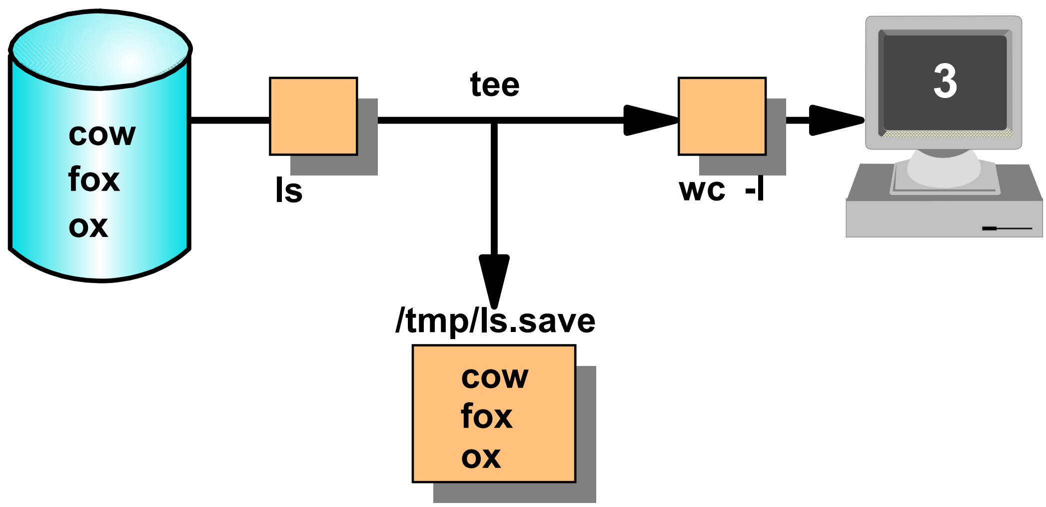


Figure 8-16. Split Outputs AU139.0

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.

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
```


Figure 8-17. Command Grouping AU139.0

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.

Line Continuation

The backslash (\) 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
```


Figure 8-18. Line Continuation AU139.0

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.

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):

Figure 8-19. Checkpoint (1 of 2) AU139.0

Notes:

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).

Figure 8-20. Checkpoint (2 of 2) AU139.0

Notes:

Exercise: Shell Basics

Figure 8-21. Exercise: Shell Basics AU139.0

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*.

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

Figure 8-22. Unit Summary AU139.0

Notes:

Unit 9. Using Shell Variables

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

Figure 9-1. Unit Objectives AU139.0

Notes:

Shell Variables

- **Variables** represent data whose value may change
- **Shell variables** define your **environment**:
  - **HOME** Directory (such as **/home/team01**)
  - **TERM**inal Type (such as ibm3151)
  - Search **PATH** (such as **/bin:/usr/bin:/etc:**.)
- **Additional variables** can be defined

Figure 9-2. Shell Variables AU139.0

Notes:

Shell variable conventions

All shell variable names are case-sensitive. For example, `HOME` and `home` are not the same.

As a convention, upper case names are used for the standard variables set by the system and lower case 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.

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
$ _
```


Figure 9-3. Listing Variable Settings AU139.0

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.

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

$
```


Figure 9-4. Setting and Referencing Shell Variables AU139.0

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).

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
```


Figure 9-5. Shell Variables Example AU139.0

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.

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
```


Figure 9-6. Command Substitution AU139.0

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.

Quoting Metacharacters

' ' Single Quotes:
 	
`$ echo '$HOME'`
`$HOME`
	Ignores all metacharacters between the quotes

" " Double Quotes:
 	
`$ echo "$HOME"`
`/home/team01`
	Ignores all metacharacters except for **dollar $, backquotes ` and backslash **

\\ Backslash:
 	
`$ echo \$HOME`
`$HOME`
	Ignores the special meaning of the following character

Figure 9-7. Quoting Metacharacters AU139.0

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 "
```

Command Line Parsing

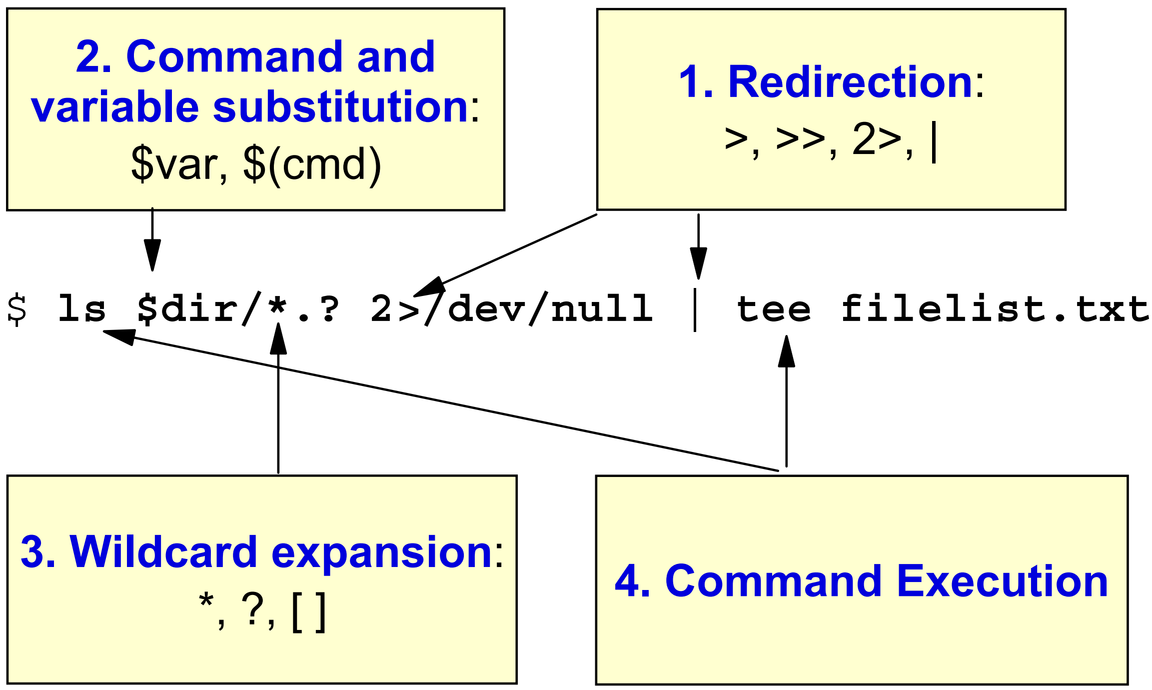


Figure 9-8. Command Line Parsing AU139.0

Notes:

Command line parsing options

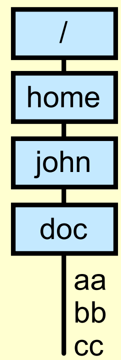
When the shell parses a command line, it is breaking the line into a series of words. One of these words determines which command will execute. Other words are information passed to the commands such as file names and options. Some of the words are instructions to the shell, like redirection.

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.

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'`

Figure 9-9. Checkpoint (1 of 2) AU139.0

Notes:

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 *

Figure 9-10. Checkpoint (2 of 2) AU139.0

Notes:

Exercise: Using Shell Variables

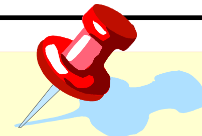
Figure 9-11. Exercise: Using Shell Variables AU139.0

Notes:

After completing the exercise, you will be able to:

- List shell built-in variables
- Use variable and command substitution
- Use quoting metacharacters.

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)

Figure 9-12. Unit Summary AU139.0

Notes:

Unit 10. Processes

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

Figure 10-1. Unit Objectives AU139.0

Notes:

What Is a Process?

- Each program runs in a process:

The Process Environment	
Program	User and group id
Data	Process ID (PID)
Open files	Parent Process ID (PPID)
Current directory	Program variables

- The variable `$$` shows the process ID of the current shell:

```
$ echo $$
4712
```

- The `ps` command shows the running processes:

```
$ ps -u team01
```


Figure 10-2. What Is a Process? AU139.0

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.

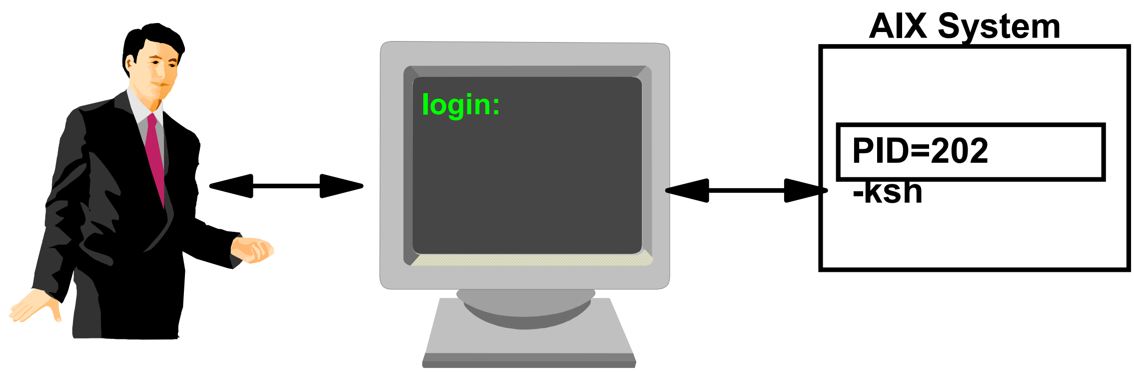
The process ID (PID) is extracted from a process table.

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**.

Login Process Environment



login: **john**
john's Password:**xxxxxx**

$ _

Environment	
program	**/usr/bin/ksh**
uid	**john**
gid	**staff**
files	**/dev/tty1**
PID	**202**

Figure 10-3. Login Process Environment AU139.0

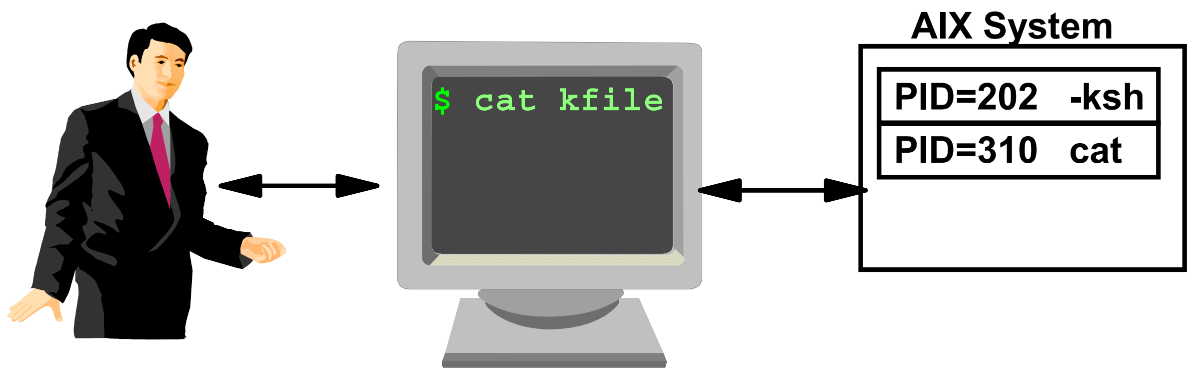
Notes:

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.

Process Environment



Environment during execution

program	**/usr/bin/cat**
uid	**john**
gid	**staff**
files	**/dev/tty1**
	kfile
parent	**-ksh**
PID	**310**
PPID	**202**

Figure 10-4. Process Environment AU139.0

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 into 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.

Parents and Children

```
$  echo $$
202

$  ksh                         (Create a subshell)
```

Subshell

```
$  echo $$
206

$  date                    (Run a command)
Tue  Jan 4  11:18:26  GMT  2000

$  <ctrl-d>                    (Exit the subshell)
```

```
$  echo $$
202
```

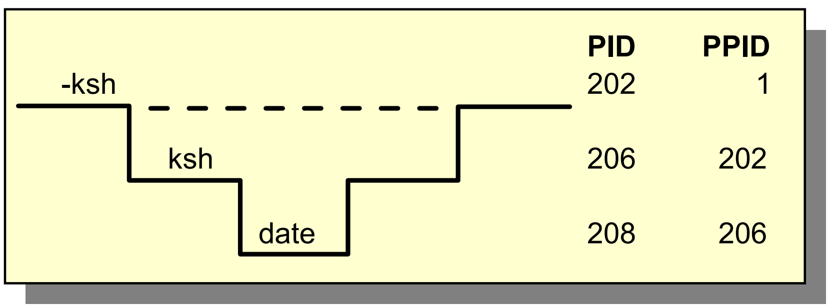


Figure 10-5. Parents and Children AU139.0

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 doesn't 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.

Variables and Processes

- **Variables** are part of the process **environment**
- Processes **cannot access or change variables** from another process

Subshell

```
$ x=4
$ ksh
$ echo $x

$ x=1
$ <ctrl-d>
$ echo $x
4
```


Figure 10-6. Variables and Processes AU139.0

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's shown in the next activity.

Activity: Exporting Variables

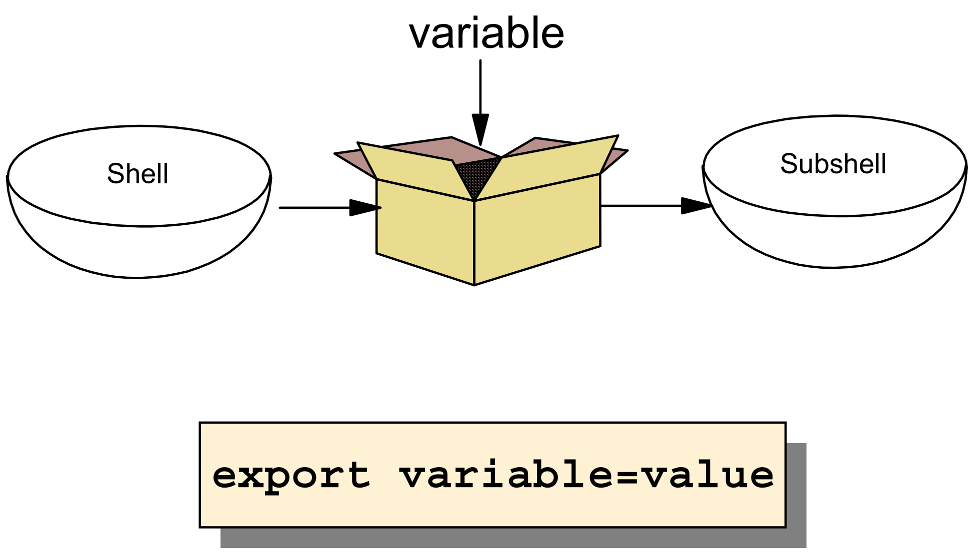


Figure 10-7. Activity: Exporting Variables AU139.0

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.

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

$
```


Figure 10-8. What Is a Shell Script? AU139.0

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 5L V5.3 online documentation using search arguments such as korn shell, **`ksh`** & programming.

Invoking Shell Scripts (1 of 3)

```
$  cat hello
echo "Hello, John. Today is: $(date)"          (1)
pwd                                             (2)
ls                                              (3)

$ ksh hello
```

Subshell

```
Hello, John: Today is: Wed Sep 13 19:34         (1)
/home/john                                      (2)
books   letter1   text2sarah                    (3)

$ _
```

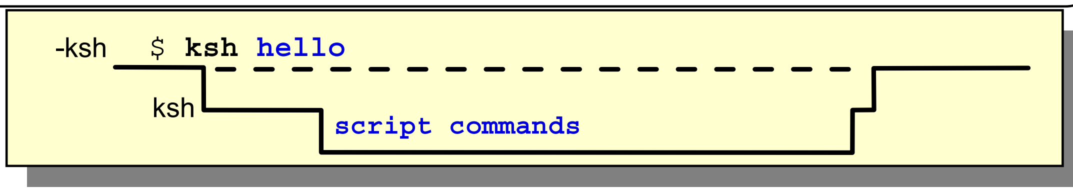


Figure 10-9. Invoking Shell Scripts (1 of 3) AU139.0

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.

Invoking Shell Scripts (2 of 3)

```
$ cat hello
echo "Hello, John. Today is: $(date)"        (1)
pwd                                          (2)
ls                                           (3)
$ chmod +x hello

$ hello
                                        Subshell
Hello, John: Today is: Wed Sep 13 19:34      (1)
/home/john                                   (2)
books  letter1  text2sarah                   (3)
$ _
```

The shell uses the PATH variable to find executable programs.

Figure 10-10. Invoking Shell Scripts (2 of 3) AU139.0

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.

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?

Figure 10-11. Invoking Shell Scripts (3 of 3) AU139.0

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.

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
```


Figure 10-12. Exit Codes from Commands AU139.0

Notes:

Process exit codes

When a command executes, if it completes successfully, it returns to it's 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.

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?)
```


Figure 10-13. Checkpoint AU139.0

Notes:

Activity: Shell Scripts

Figure 10-14. Activity: Shell Scripts AU139.0

Notes:

Activity

__ 1. Log into 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
```

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

Figure 10-15. Unit Summary AU139.0

Notes:

Unit 11. Controlling Processes

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

Figure 11-1. Unit Objectives AU139.0

Notes:

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
```


Figure 11-2. Monitoring Processes AU139.0

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.

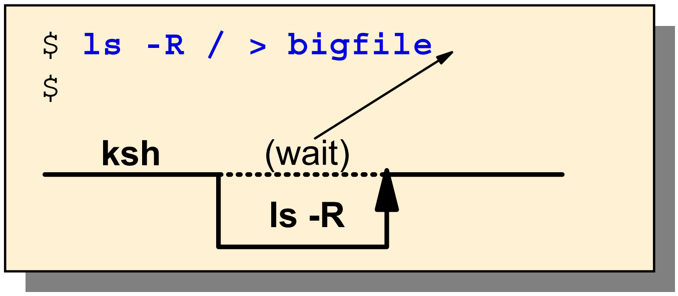
The `-f` option in addition to the default information provided by `ps`, displays the User Name, PPID, start time for each process (that is, a FULL listing).

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 5L V5.3 has support for all of the System V options.

Controlling Processes

Foreground Processes:



Background Processes (&):

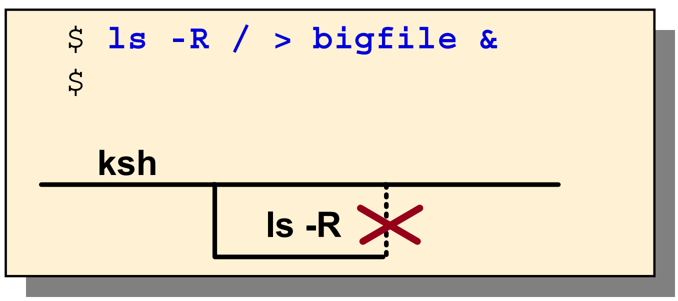


Figure 11-3. Controlling Processes AU139.0

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*. 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 doesn't require keyboard input.

ii. It is invoked with an ampersand **&** as the last character in the command line

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

Figure 11-4. Terminating Processes (1 of 2) AU139.0

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.

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!**)

Figure 11-5. Terminating Processes (2 of 2) AU139.0

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, which might cause big problems. 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).

If a simple **kill <pid>** command fails to end the process, sometimes a **kill -9** (sending a termination signal) is required to kill the does not allow the application to close down in an orderly manner.

Signals

Signal	Meaning
01	hangup - you logged out while the process was still running
02	interrupt - you pressed the interrupt (break) key sequence `<Ctrl+c>`
03	quit - you pressed the quit key sequence `<Ctrl+\>`
09	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

Figure 11-6. Signals AU139.0

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 (01) is sent to a process if its parent dies, for example if you log off when a background process is running.

The INTerrupt signal (02) is generated when the user presses the interrupt key (**Ctrl-c**) on the keyboard.

The QUIT signal (03) is generated by the user pressing the quit key **<Ctrl+\>**. Again, this is in different places on different systems.

The most powerful signal you can send to a process is a signal 09, 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.

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
$
```


Figure 11-7. Running Long Processes AU139.0

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 `01` and `03` (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**.

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.

Job Control in the Korn Shell

jobs	Lists all jobs running in the background and stopped processes
<Ctrl+z>	Suspends foreground task
fg %<jobnumber>	Executes job in foreground
bg %jobnumber	Executes job in background

Figure 11-8. Job Control in the Korn Shell AU139.0

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 labelled as job number one.

Moving a foreground process to the background

You can stop a foreground process by pressing **`<ctrl-z>`**. This does not terminate the process; it suspends it so that you can subsequently restart it.

To restart 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 into 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, starting with AIX 5L V5.3, 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 &
```

No, you want to log off, but require that the job to complete. You can 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
```

You can now log off and the job will finish.

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 &
$
```


Figure 11-9. Job Control Example AU139.0

Notes:

Introduction

This visual shows how you can work with job control commands in a Korn shell.

Daemons

A **daemon** is a never-ending process, that controls a system resource such as the printer queue

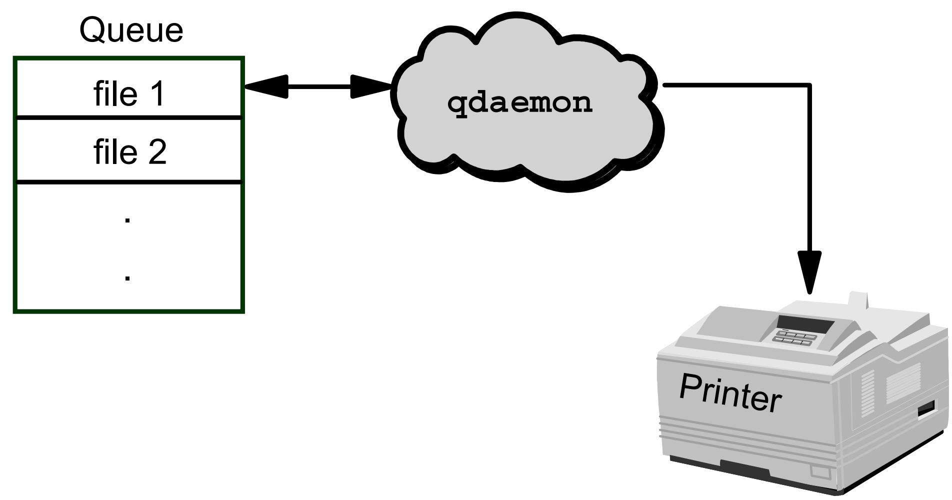


Figure 11-10. Daemons AU139.0

Notes:

Introduction

A *daemon* 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, especially for communications software.

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?

Figure 11-11. Checkpoint AU139.0

Notes:

Exercise: Controlling Processes

Figure 11-12. Exercise: Controlling Processes AU139.0

Notes:

After completing the lab, you will be able to:

- Monitor processes by using the `ps` or `jobs` command.
- Control processes.

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

Figure 11-13. Unit Summary AU139.0

Notes:

Unit 12. Customizing the User Environment

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

Figure 12-1. Unit Objectives AU139.0

Notes:

Login Files

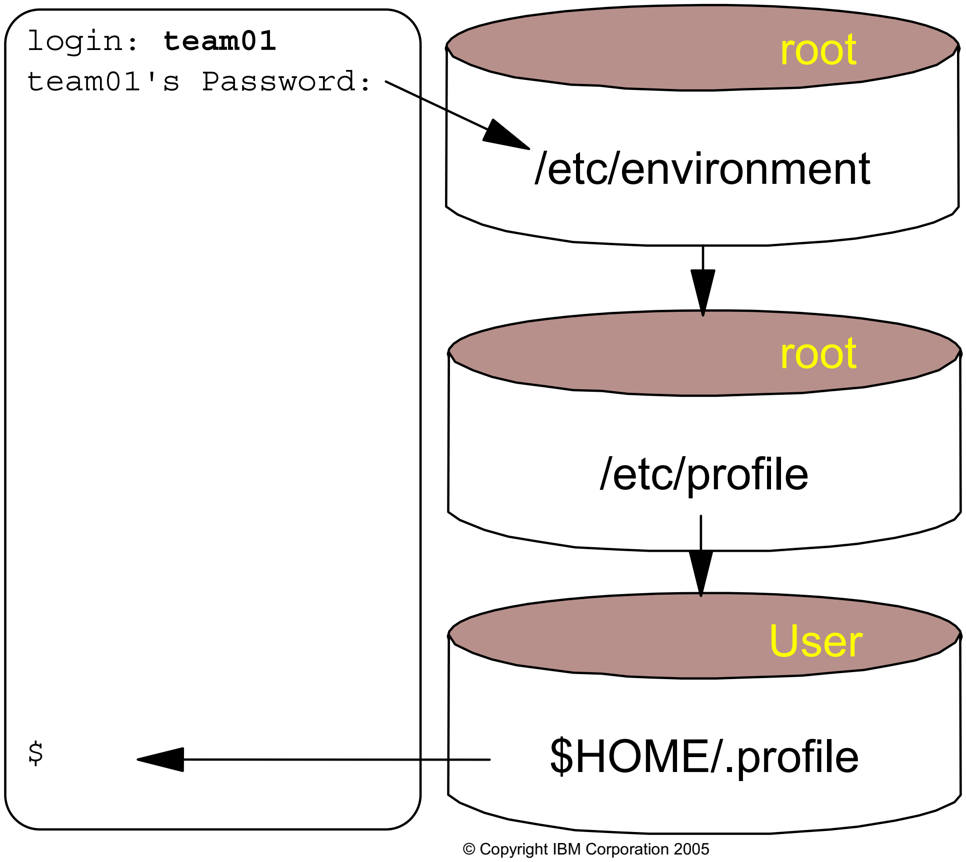


Figure 12-2. Login Files AU139.0

Notes:

Introduction

When you first log into an AIX 5L 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 file controls system-wide default variables such as the mail messages and terminal types. /**etc**/**profile** 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.

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
```


Figure 12-3. Sample **/etc/environment** AU139.0

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.

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
```


Figure 12-4. Sample **/etc/profile** AU139.0

Notes:

/etc/profile

The **/etc/profile** file contains the set of environment variables and commands that will be invoked when a user logs into the system. These are settings that all users will have applied to their shell as they login.

Any settings here can be overridden by a user's **.profile**.

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.**

Figure 12-5. Environment Variables (1 of 2) AU139.0

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.

Sample .profile

```
$ cat .profile

PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:$HOME/bin:/usr/bin/X11:/sbin:.

PS1=' $PWD => '

ENV="$HOME/.kshrc"

export PATH PS1 ENV

if [  -s  "$MAIL"   ]
then
     echo  "$MAILMSG"

fi
```

Execute this file every time a new Korn shell is started.

Figure 12-6. Sample **.profile** AU139.0

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.

.profile and Common Desktop Environment

Be aware that your **.profile** file may not be read if you are accessing the system through Common Desktop Environment (CDE). By default, CDE instead uses a file called **.dtprofile**. In the CDE environment, if you wish to use the **.profile** file, it is necessary to uncomment the `DTSOURCEPROFILE` variable assignment at the end of the **.dtprofile** file.

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"`

Figure 12-7. Environment Variables (2 of 2) AU139.0

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.

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  ..'
```


Figure 12-8. Sample **.kshrc** AU139.0

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.

`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 -'
```


Figure 12-9. `ksh` Features - Aliases AU139.0

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 muonic. 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 reexecute the previously entered command.

Passing aliases to subshells

To carry down the value of an alias to subsequent 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.

ksh Features - Using Aliases

```
$ ll              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  ll     Remove alias definition


$ ll
ksh:  ll:   not found
```


Figure 12-10. ksh Features - Using Aliases AU139.0

Notes:

Using aliases

To use an alias, simply invoke the alias name as if it where 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.

ksh Features - History

Last 128 commands are stored in file **$HOME/.sh_history**

$ history

```
$  fc -l
2  cd /home/payroll
3  ls -l
4  mail
5  fc -l


$ 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
```


Figure 12-11. ksh Features - History AU139.0

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 -l` command reads this file and allows you to list the last 16 commands entered. Instead of `fc -l` 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.

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:`

Figure 12-12. Checkpoint AU139.0

Notes:

Exercise: Customizing the User Environment

Figure 12-13. Exercise: Customizing the User Environment AU139.0

Notes:

After completing the exercise, you will be able to:

- Customize **.profile** and **.kshrc** files.
- Set `alias` definitions.

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

Figure 12-14. Unit Summary AU139.0

Notes:

Unit 13. AIX Utilities

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 **dosread**, **doswrite**, **dosdel** and **dosformat** commands to manipulate files from a PC-DOS environment
- Use the **tn** 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 **`dosread`**, **`doswrite`**, **`dosdel`**, **`dosdir`**, and **`dosformat`** commands to manipulate files from a PC-DOS environment
- Use the **`tn`** and **`ftp`** commands to communicate with other hosts

Figure 13-1. Unit Objectives AU139.0

Notes:

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
```


Figure 13-2. `find` AU139.0

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.>`.

Sample Directory Structure

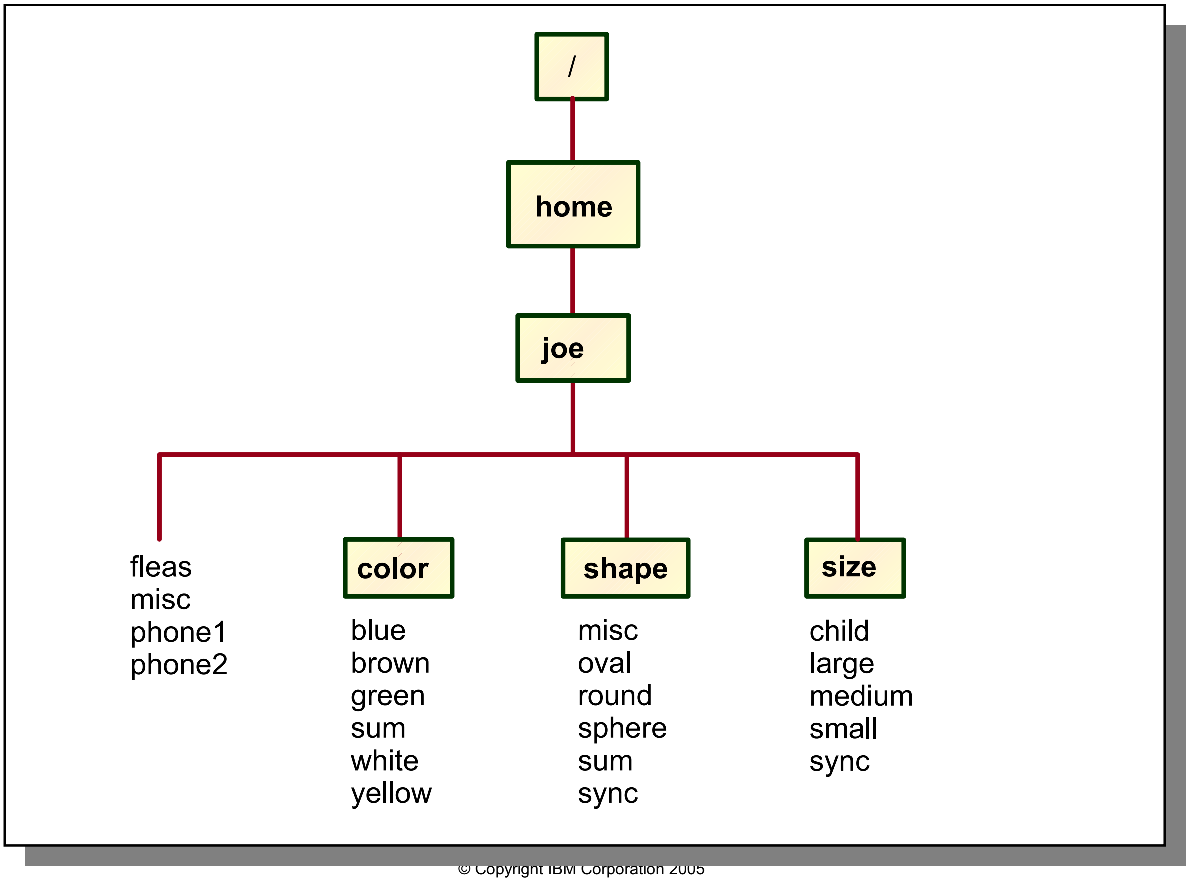


Figure 13-3. Sample Directory Structure AU139.0

Notes:

`find` examples

This sample directory structure will be used in the examples on the following pages.

Using `find`

- Search a directory structure for files with certain names:

```
$ find . -name sum
 ./color/sum
 ./shape/sum
```

- On many UNIX systems, you must use `-print`:

```
$ find . -name sum -print
 ./color/sum
 ./shape/sum
```


Figure 13-4. Using `find` AU139.0

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.

Executing Commands with `find`

The **exec** option executes a command on each of the files found

Matched file name

```
$ 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
```


Figure 13-5. Executing Commands with **find** AU139.0

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
```

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
```


Figure 13-6. Interactive Command Execution AU139.0

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.

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" access permissions match "mode" values (ex. rwx)
`-user`	user	finds files owned by "user"
`-o`		logical "or"
`-newer`	ref.file	searches for files that are newer than the reference file (you specify the reference file)

Figure 13-7. Additional Options AU139.0

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.

The Shell versus `find`

Scenario: Starting at current directory find all files that start with c

```
$ ls c*
c1 c2

$ find . -name 'c*'
./c1
./c2
./dir1/c3
./dir1/c4
./dir1/dir2/c5
./dir1/dir2/c6
```

Shell expands wildcard

find expands wildcard

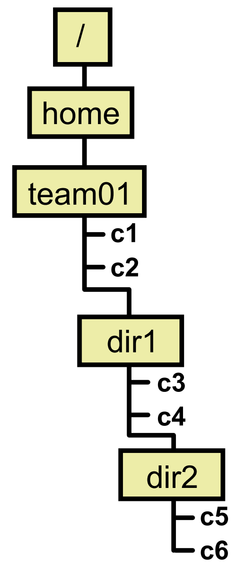


Figure 13-8. The Shell versus `find` AU139.0

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*
```


`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
```


Figure 13-9. `find` Examples AU139.0

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**.

Exercise: AIX Utilities (1)

Figure 13-10. AIX Utilities (1) AU139.0

Notes:

After completing the exercise, you will be able to:

- Use the `find` command to find files that meet specific criteria.

grep

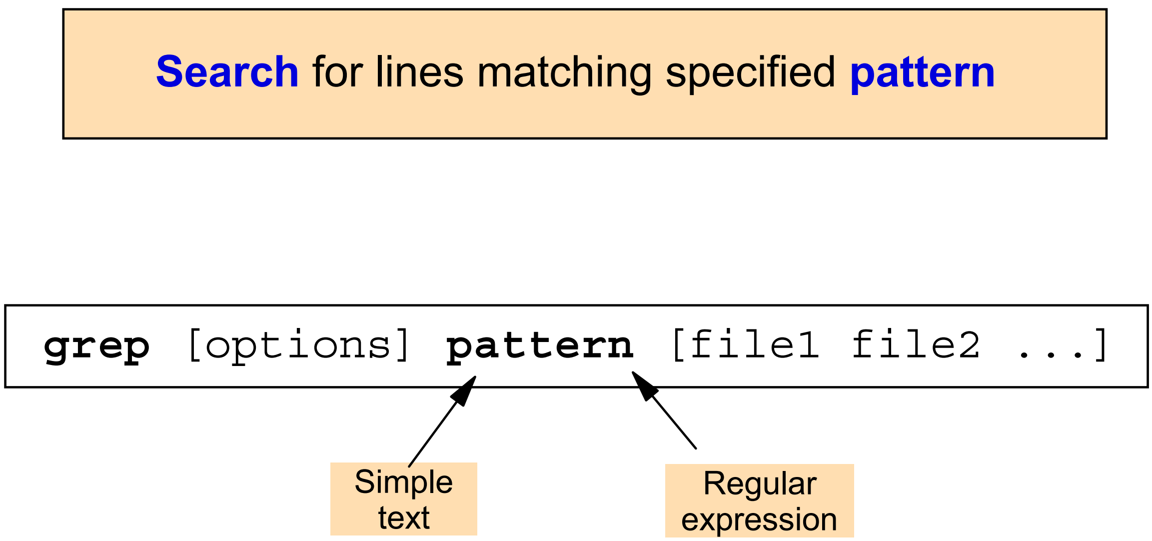


Figure 13-11. `grep` AU139.0

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 upper case T.

The command displays the name of the file containing the matched line, if more than one file is specified for the search.

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
```


Figure 13-12. grep Sample Data Files AU139.0

Notes:

grep examples

This visual shows the sample files used to illustrate the examples of grep that follow on the next visual.

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
```


Figure 13-13. Basic grep AU139.0

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.

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

Figure 13-14. grep with Regular Expressions AU139.0

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.

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
```


Figure 13-15. grep Examples AU139.0

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.

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

Figure 13-16. grep Options AU139.0

Notes:

grep Options

The visual shows different grep options.

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
```


Figure 13-17. Other **grep** Commands. AU139.0

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
```

Activity: `grep` Command

Figure 13-18. Activity: `grep` Command AU139.0

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**.

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.

Figure 13-19. sort Command AU139.0

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.

sort Examples

```
$ cat animals
dog.2
cat.4
elephant.10
rabbit.7

$ sort animals
cat.4
dog.2
elephant.10
rabbit.7

$ cat animals | sort +0.1
rabbit.7
cat.4
elephant.10
dog.2

$ cat animals | sort -t. -n +1
dog.2
cat.4
rabbit.7
elephant.10
```

Default sort order

Sort by second character

-t: Delimiter "."
-n: Numerical order
+1: Second field

Figure 13-20. sort Examples AU139.0

Notes:

Examples

This visual shows different ways the sort command can be used.

`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
```


Figure 13-21. `head` and `tail` Commands AU139.0

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. 10 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.

Transferring DOS Data Files

`$ dosdir -l`	**List** the contents of a DOS diskette
`$ dosread file1.doc file1`	**Copy a file** from diskette **to AIX**
`$ doswrite file1 file1.doc`	**Copy a file** from AIX **to a DOS** diskette
`$ dosread -a letter.txt letter` `$ doswrite -a letter letter.txt`	**Convert** an AIX text file to DOS format or a DOS text file to AIX format
`$ dosdel filez`	**Delete** a file from a DOS diskette

Figure 13-22. Transferring DOS Data Files AU139.0

Notes:

DOS file utilities

The default DOS device is the first diskette drive, **/dev/fd0**. The default location for AIX files is the current directory.

To specify a DOS device other than the top diskette drive use the `-D` option followed by the name of the device.

To specify a pathname on the DOS diskette use the forward slash (/) rather than the backslash which DOS normally uses.

The `dosdir` command lists the files on the diskette. The `-l` option also lists the sizes and modification times for the files.

If no destination file is specified for the `dosread` command, then the file is written to standard output.

The `-a` option converts the character sequence CRLF (carriage return-line feed) to a NL (new line) character and interprets a `<Ctrl-z>` as EOF (end-of-file) when reading a DOS file. The reverse translation is performed when writing a DOS file.

If you do not use the `-a` option, a binary copy is done and no control codes are converted.

To delete a file from a DOS diskette use the **dosdel** command.

To format a DOS diskette use the **dosformat** command. Note that there is another command to format AIX diskettes - the **format** command.

`tn`: Login to Remote Hosts

Use the `tn`-command to login to remote hosts

Example:

```
$ tn miami
Trying ...
Connected to miami
...






AIX Version 5
(C) Copyright by IBM and others 1982, 1996
login: team01
```


Figure 13-23. `tn`: Login to Remote Hosts AU139.0

Notes:

`tn` command

The `tn` (`telnet`) command allows a user to login 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.

`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

Figure 13-24. `ftp`: Transfers Files Between Hosts AU139.0

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.

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
```


Figure 13-25. ftp Subcommands AU139.0

Notes:

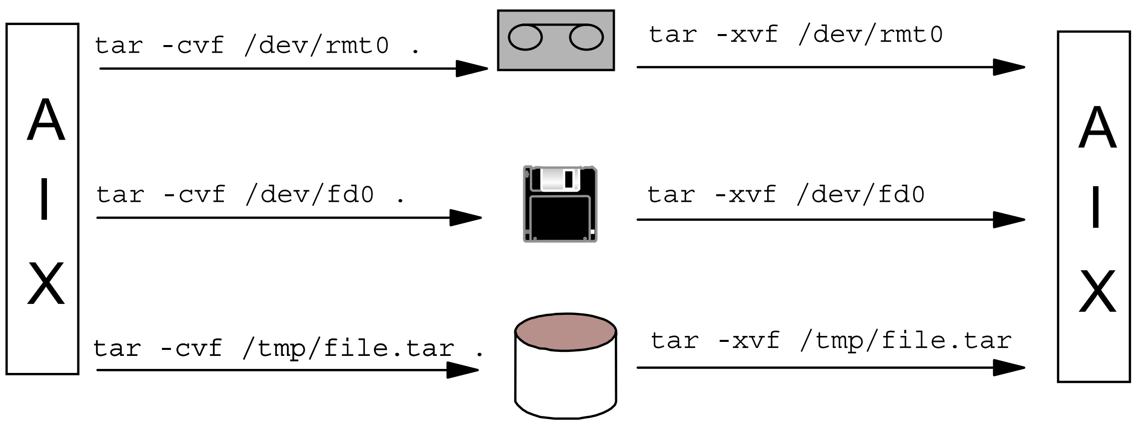
Controlling ftp

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
```

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)`

Figure 13-26. `tar`: Backup and Restore Files AU139.0

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

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`

Figure 13-27. Checkpoint AU139.0

Notes:

Exercise: AIX Utilities (2)

Figure 13-28. Exercise: AIX Utilities (2) AU139.0

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.

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`**

Figure 13-29. Unit Summary AU139.0

Notes:

Unit 14. AIX Utilities, Part II

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 `dircmp` 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 **`dircmp`** to compare files and directories
- **Compress files** to save space
- Display **non-printable** characters in files and directories

Figure 14-1. Unit Objectives AU139.0

Notes:

xargs

Reads a **group of arguments from stdin;** runs an AIX command with that group of arguments:

```
$ cat oldfilelist
file1
file2                    file1 file2 file3 file4
file3
file4

$ cat oldfilelist | xargs -t rm
rm file1 file2 file3 file4
```


Figure 14-2. **xargs** AU139.0

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.

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
```


Figure 14-3. xargs Examples AU139.0

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 don't 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.

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
```


Figure 14-4. xargs, find, and grep AU139.0

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**.

The `-links` Option with `find`

```
$ find /home -type f -links +1 | xargs ls -li

127 -rw-r--r-- 3 team01 staff 156 July 26 13:11 /home/team01/myfile
127 -rw-r--r-- 3 team01 staff 156 July 26 13:11 /home/team01/yourfile
127 -rw-r--r-- 3 team01 staff 156 July 26 13:11 /home/team02/akafile
```

Same i-nodes

Number of links

Same number of bytes

Figure 14-5. The `-links` Option with `find` AU139.0

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.

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
```


Figure 14-6. `alias` and `find` AU139.0

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.

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
```


Figure 14-7. which, whereis, and whence AU139.0

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.

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
```


Figure 14-8. `file` AU139.0

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 files that have some sort of magic number, that is, any file containing a numeric or string constant that indicates the type.

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.

Exercise: AIX Utilities (3)

Figure 14-9. Exercise: AIX Utilities (3) AU139.0

Notes:

After completing the exercise, you will be able to:

- Use the `find`, `xargs`, and `file` commands.

`diff` (Differential File Comparator)

- Analyzes text files
- Reports the **differences** between files

```
diff [-options] file1 file2
```


Figure 14-10. `diff` (Differential File Comparator) AU139.0

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.

Comparing Two Files Using `diff`

names: (<)

Carol
Bob
Jim
Don
Mary

names.old: (>)

Caroline
Bob
Jim
Mary
Sam

```
$ diff names names.old

1c1
< Carol
----
> Caroline
4d3
< Don
5a5
> Sam
```


Figure 14-11. Comparing Two Files Using `diff` AU139.0

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.

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
```


Figure 14-12. Comparing Two Files Using `cmp` AU139.0

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.

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                          ./b1
./dir1/c3
./dir1/c4
./dir1/dir2
./dir1/dir2/c5
./dir1/dir2/c6

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

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
```

1: List files **unique** to each directory

2: List files with **identical** names

3: Display **differences** for common files

Figure 14-13. Comparing Directories Using `dircmp` AU139.0

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

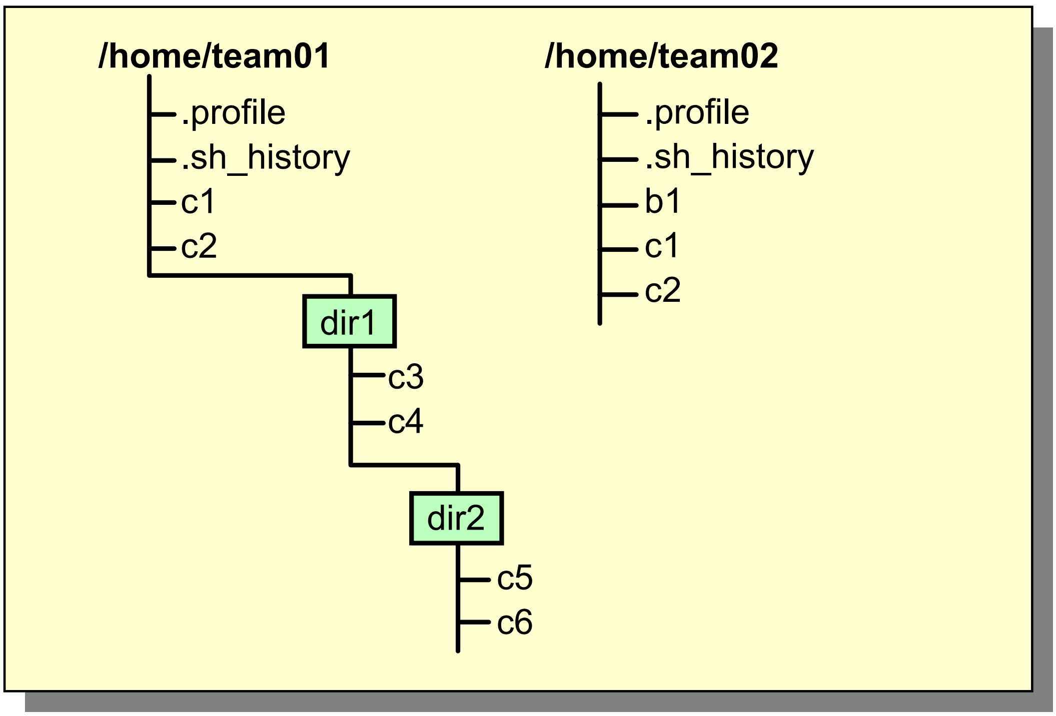
The `-d` option lists the `diff` output which is displayed last. The `-s` option could have been listed to silence or not display the files that are named the same and have identical contents, as indicated by the word same in the second area of information.

Command output

Be sure to pipe the output of the `dircmp` command to `pg` or `more` as it will produce multiple pages of output.

Directory structure used in the examples

Sample Directory Structure



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
```


Figure 14-14. compress, uncompress, and zcat AU139.0

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 compress version without the **.Z** extension.

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
```


Figure 14-15. Displaying Non-Printable Characters in Files AU139.0

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.

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 {} \;
```


Figure 14-16. Non-Printable Characters in Directories AU139.0

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 don't 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 can't remove the file using method 1, find the i-node number of the file and use the `find` command with the `-inum` expression

Assigning Unique File Names

```
$ touch myfile$$
$ 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 process ID: **$$**

Append the date using a command substitution

Figure 14-17. Assigning Unique File Names AU139.0

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.

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`
   `dircmp`
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

Figure 14-18. Checkpoint AU139.0

Notes:

Exercise: AIX Utilities (4)

Figure 14-19. Exercise: AIX Utilities (4) AU139.0

Notes:

After completing the exercise, you will be able to:

- Use `diff`, `cmp` and `dircmp` to compare files and directories.
- Use `compress`, `zcat` and `uncompress`.
- Use `cat` to display non-printable characters.

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
- **`dircmp`** is used to compare the contents of two directories
- **`compress`** compresses data in files using Lempel-Zev coding

Figure 14-20. Unit Summary AU139.0

Notes:

Unit 15. Additional Shell Features

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

Figure 15-1. Unit Objectives AU139.0

Notes:

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

Figure 15-2. Important Shell Variables AU139.0

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.

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
```


Figure 15-3. Positional Parameters AU139.0

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.

The expr Utility

- Use the **expr** utility to perform integer arithmetic

- **expr** offers the following **operators**:

```
\*    multiplication
/     integer division
%     remainder
+     addition
-     subtraction (also unary minus sign)
```

Precedence

high

low

Figure 15-4. The expr Utility AU139.0

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`.

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
```


Figure 15-5. expr Examples AU139.0

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.

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
```


Figure 15-6. Conditional Execution AU139.0

Notes:

Conditional examples

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.

`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:
`$string1 = $string2`	Strings are identical
`$string1 != $string2`	Strings are not identical
`$number1 -eq $number2`	Numbers are equal
`$number1 -ne $number2`	Numbers are not equal
`-e $file`	File exists
`-d $file`	File is a directory
`-r $file`	File is readable
`-w $file`	File is writable

Figure 15-7. `test` Command AU139.0

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.

`if` Command

```
if    condition is true
then
      carry out this set of actions
else
      carry out these alternative actions    } optional
fi
```

```
$ 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
```


Figure 15-8. `if` Command AU139.0

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.

Activity: Writing Shell Scripts

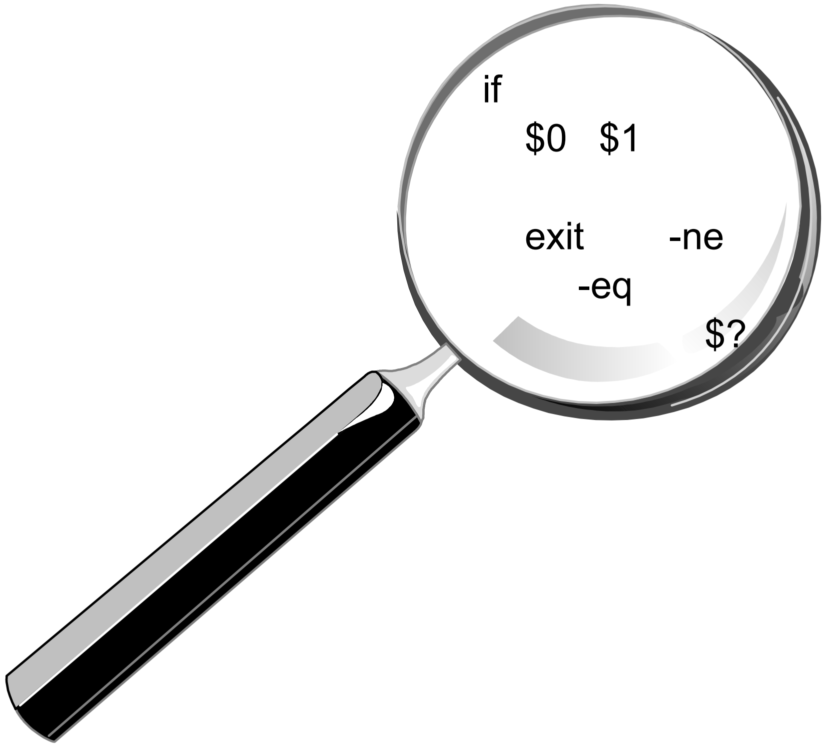


Figure 15-9. Activity: Writing Shell Scripts AU139.0

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 *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**.

`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
```


Figure 15-10. `read` Command AU139.0

Notes:

Using the `read` command

The **`read`** command can be used to assign more than one variable value. If more than one argument is given with the script, for instance in this example if **`delfile`** was invoked with more than one file name, the first file name would be assigned to the first variable name defined by the **`read`** statement, the second file name 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.

for Loop Syntax

for variable **in** list
do
command(s)
done

```
$ cat count
for var in file1 file2 file3
do
  wc -l $var
done

$ count
  18  file1
  20  file2
  12  file3
```

```
$ cat rm_tmp
for FILE in /tmp/*
do
  echo "Removing $file"
  rm $FILE
done
```


Figure 15-11. `for` Loop Syntax AU139.0

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.

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
```


Figure 15-12. `while` Loop Syntax AU139.0

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.

Command Search Order

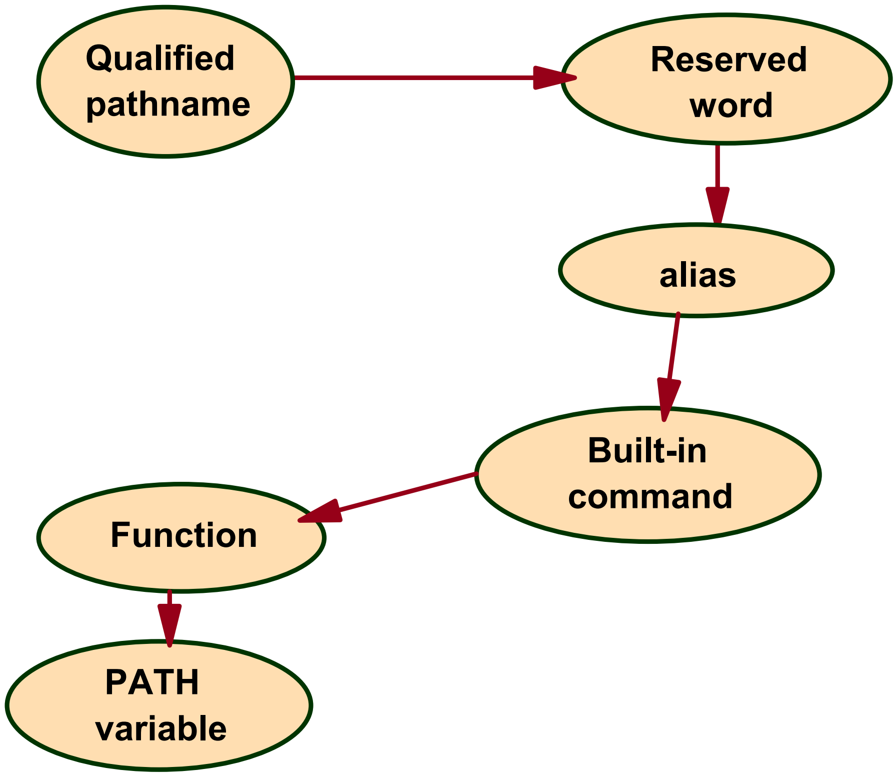


Figure 15-13. Command Search Order AU139.0

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 5L 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.

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
```


Figure 15-14. Sample **.profile** AU139.0

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.

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 that will multiply any two numbers together.

Figure 15-15. Checkpoint AU139.0

Notes:

Exercise: Additional Shell Features

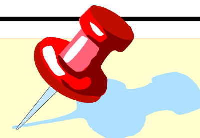
Figure 15-16. Exercise: Additional Shell Features AU139.0

Notes:

After completing the exercise, you will be able to:

- List common constructs used in writing shell scripts.
- Create and execute simple shell scripts.

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

Figure 15-17. Unit Summary AU139.0

Notes:

Unit 16. AIXwindows Concepts

What This Unit Is About

This unit provides an overview of the AIXwindows 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
- Move, resize, maximize, minimize and close an AIX window
- Start an `aixterm` window
- Display remote clients on your system

How You Will Check Your Progress

- Student Activity
- Checkpoint questions
- Exercise

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
- Move, resize, maximize, minimize and close AIX windows
- Start an aixterm window
- Display remote clients on your system

Figure 16-1. Unit Objectives AU139.0

Notes:

The Evolution of X Window

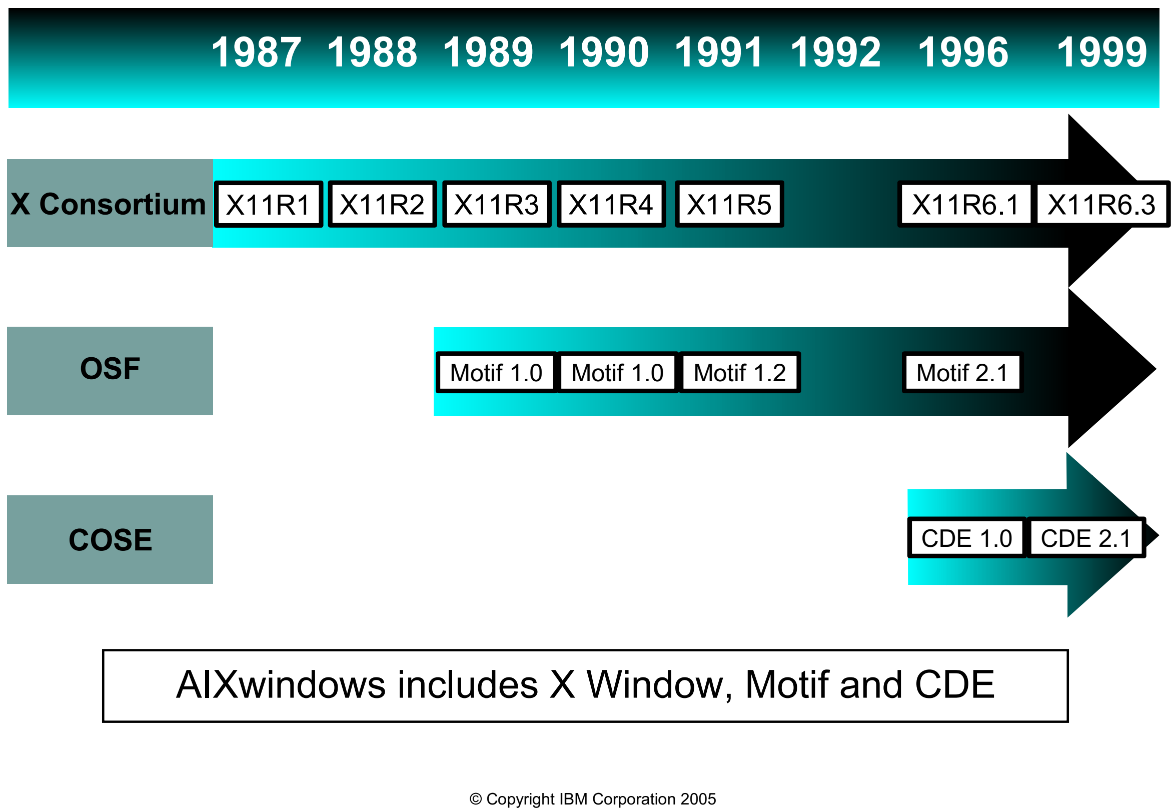


Figure 16-2. The Evolution of X Window AU139.0

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 6 (1994). The X-Consortium code is freely available and will run on most UNIX architectures. X Window improves UNIX in much the same way that MS-Windows improves PC-DOS.

AIXwindows

AIXwindows is AIX's windowing system. AIXwindows includes X Window, 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. The Common Desktop Environment (CDE) will be discussed in more detail in later units.

What is AIXwindows?

- AIXwindows is IBM's **enhancement** to **X Window** 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**

Figure 16-3. What is AIXwindows? AU139.0

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.

X Window 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.

An X Window Network Configuration

- **Networked Workstations** and File Servers
- Heterogeneous Environment
- A **Client/Server** Environment

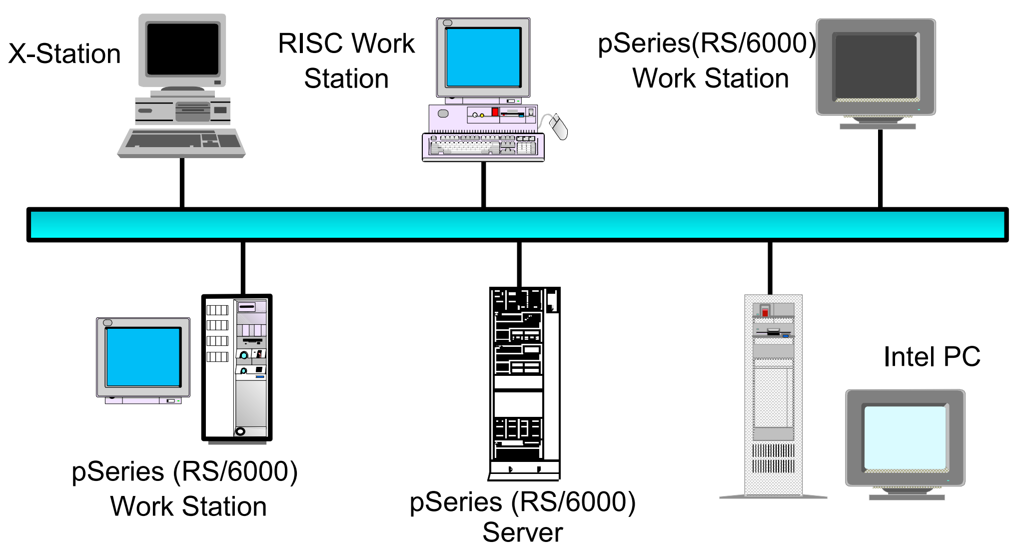


Figure 16-4. An X Windows Network Configuration AU139.0

Notes:

AIXwindows in a network environment

Above is an example of an X Window network. X Window is platform independent. X Window allows a display and keyboard attached to one system to use programs running on a completely different type of system.

X Window 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 X Window is commonly referred to as a *networking* window system.

This is another way of saying that X Window 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.

The Client/Server Environment

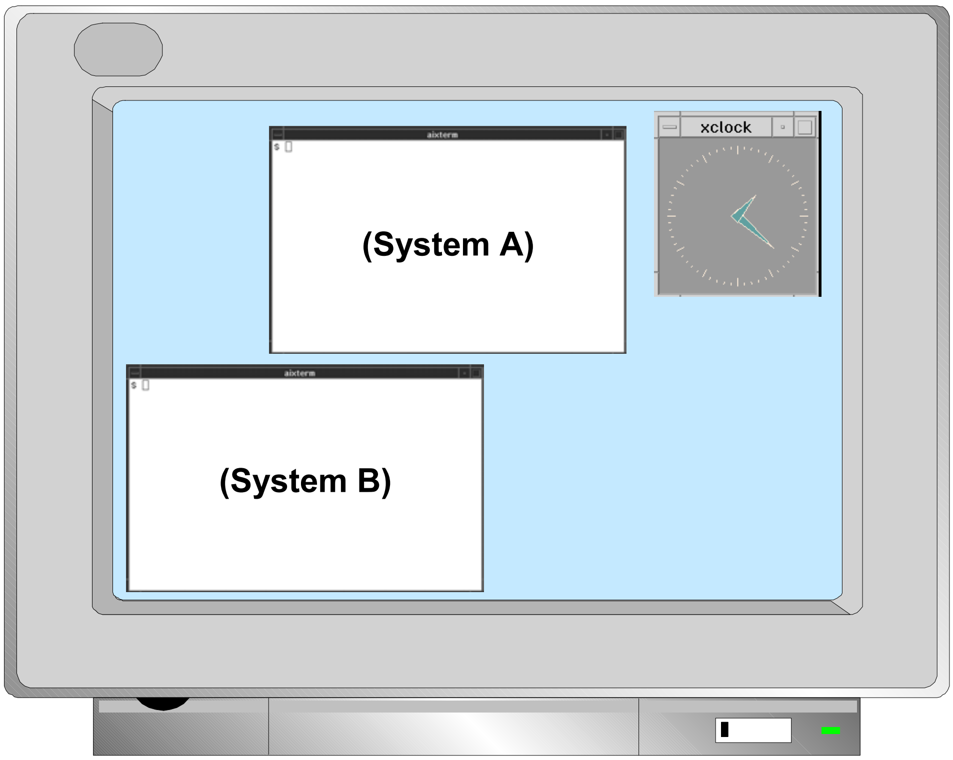


Figure 16-5. The Client/Server Environment AU139.0

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.

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

Figure 16-6. X Clients AU139.0

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 Window

`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.

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**

Figure 16-7. The X Server AU139.0

Notes:

Introduction

The X Server is a program that runs on an X Station, on a PC or on a UNIX system. The X Server is basically the only thing that runs on an X Station.

The X Server

The X Client and X Server must cooperate with each other. If the X Client uses a font that the X Server does not have, then you will get an error message.

In the X world, the most common place for the X Server to run is probably on the same system as the one that the X Client runs on.

Starting AIXwindow

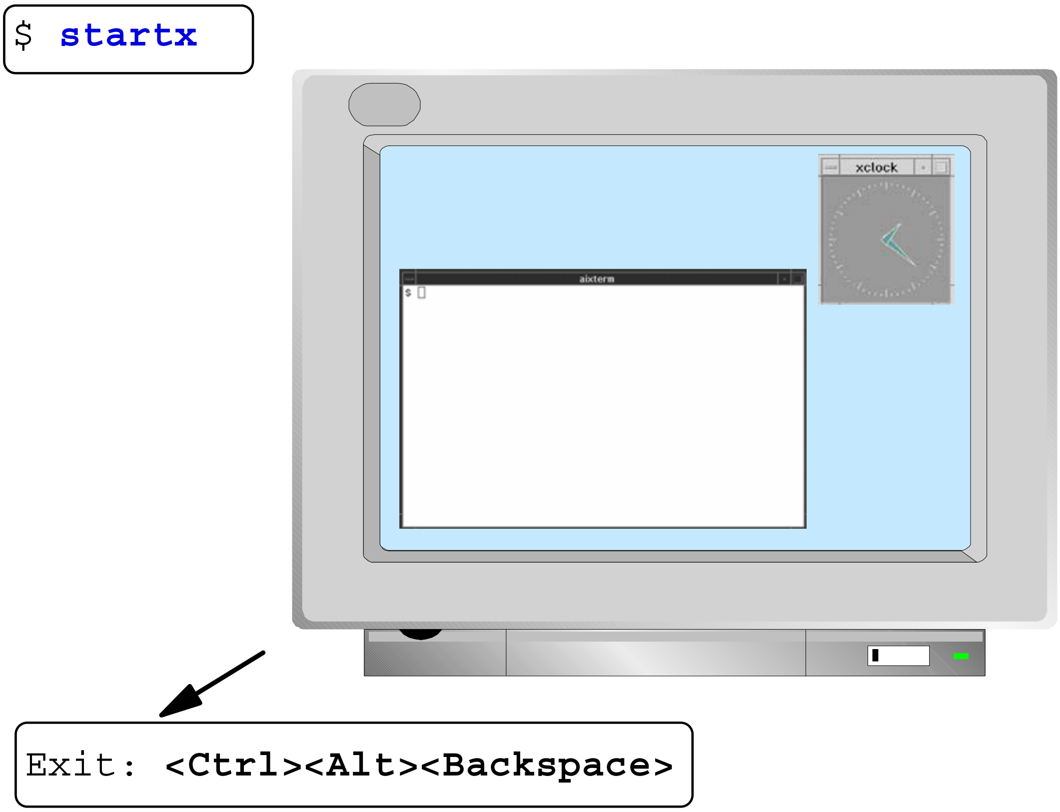


Figure 16-8. Starting AIXwindows AU139.0

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 X Station), **`startx`** will execute the **`xinit`** command.

What is started

By default, **`startx`** will also start three X Clients: an **`aixterm`**, the **`xclock`** and the **`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 start up process will be sent to a file called **$HOME/.xerrors**. If the file does not exist, it will be created automatically.

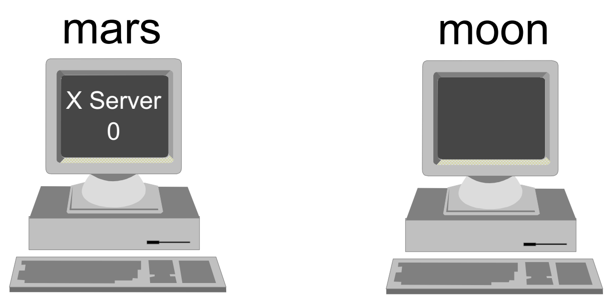
It is also possible to set an environment variable XERRORS. This variable should be set if you wish to route errors to a file other than **$HOME/.xerrors**.

Exiting AIXwindows

Use the key sequence `<Ctrl><Alt><Backspace>` to close AIXwindows and return to the command prompt.

Activity: AIXwindows Concepts

1. Which command starts **AIXwindows** and the **X Server** on system **mars**?



2. What environment variable is used by X Clients to identify the X Server to use, if no option is specified?

3. True or false: Before starting the **xterm** client on **moon**, the **`startx`** command must be executed on the client system.

4. The **`startx`** command fails. Which file describes possible error reasons?

Figure 16-9. Activity: AIXwindows Concepts AU139.0

Notes:

Activity

Take some time and try to answer the questions in the visual.

An AIXwindows Display

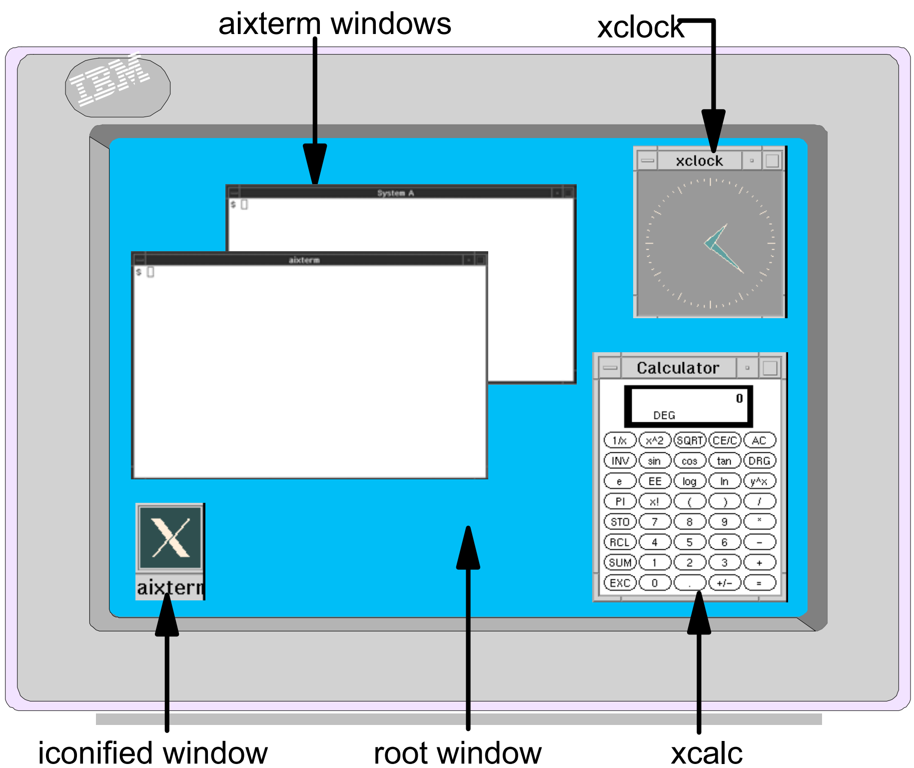


Figure 16-10. An AIXwindows Display AU139.0

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.

Input Focus

- Use the **mouse** to select the **active window** (focusing)
- Only one window can have **input focus** (active window)
- Types of focus:
  - **Explicit**: Click **active window** with **left mouse** button
  - **Pointer**: Activates window which is **under the pointer**
- Change the **default focus policy** in **$HOME/.Xdefaults:**

```
Mwm*keyboardFocusPolicy: pointer
```


Figure 16-11. Input Focus AU139.0

Notes:

AIXwindows input focus

Motif can work with either a two- or three-button mouse. Most RS/6000 systems use a three-button mouse. Moving a mouse on the screen moves a small icon called the *mouse pointer*.

The mouse pointer can be used to select the *active* window, which is the window capable of accepting input. Directing input to a particular window is called *focusing*. When a window has input focus, its window frame will be highlighted.

There are two types of focus: *explicit*, which is the Motif Window Manager default, and *pointer*. *Explicit* focus is accomplished by moving the mouse pointer to the window and clicking with the left mouse button. *Pointer* focus will activate whichever window is under the pointer.

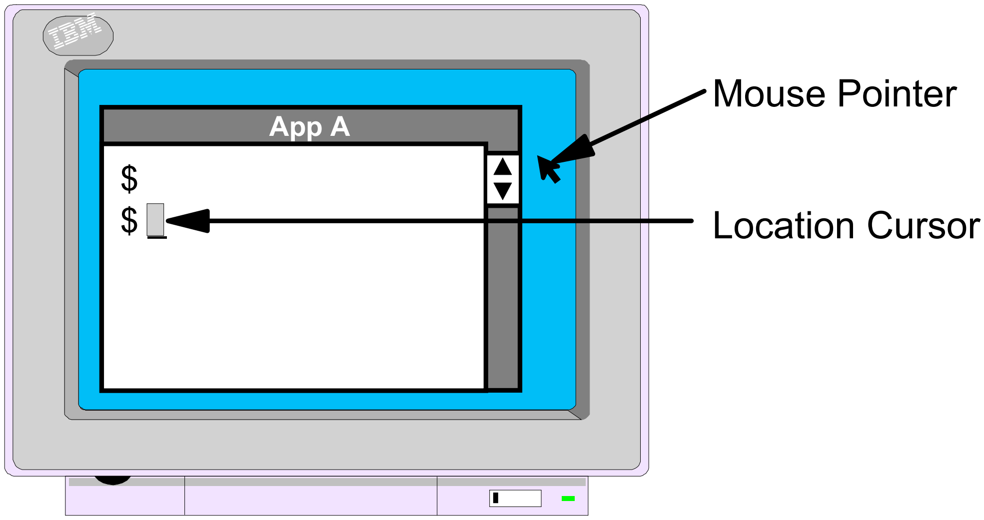
Customizing the AIXwindows environment

A user's AIXwindows environment can be customized by building a **$HOME/.Xdefaults** file. Code the following in this file to change the default focus policy:

```
Mwm*keyboardFocusPolicy:  pointer
```

The **$HOME/.Xdefaults** file will be covered in more detail in the next unit.

The Mouse Pointer and Location Cursor



- **Mouse** controls the movement of the **pointer**
- **Pointer** is used to
  - **activate menus**
  - **change window focus**
- **Location cursor** determines where **keyboard input** will appear

Figure 16-12. The Mouse Pointer and Location Cursor AU139.0

Notes:

Mouse pointer

The movement of the mouse is translated into a movement of the pointer depicted on the screen. The shape of the pointer can vary. There are several that you will see.

The pointer is placed in or on items on the screen. Pressing the left or right (or center) button of the mouse will cause some selection or action to take place.

Location cursor

The location cursor is very similar to the cursor on the command line of a character based screen. Its location determines where keyboard input will be placed in the window. It is usually necessary to give a window focus with the mouse pointer before the location cursor takes effect.

The Motif Window Frame

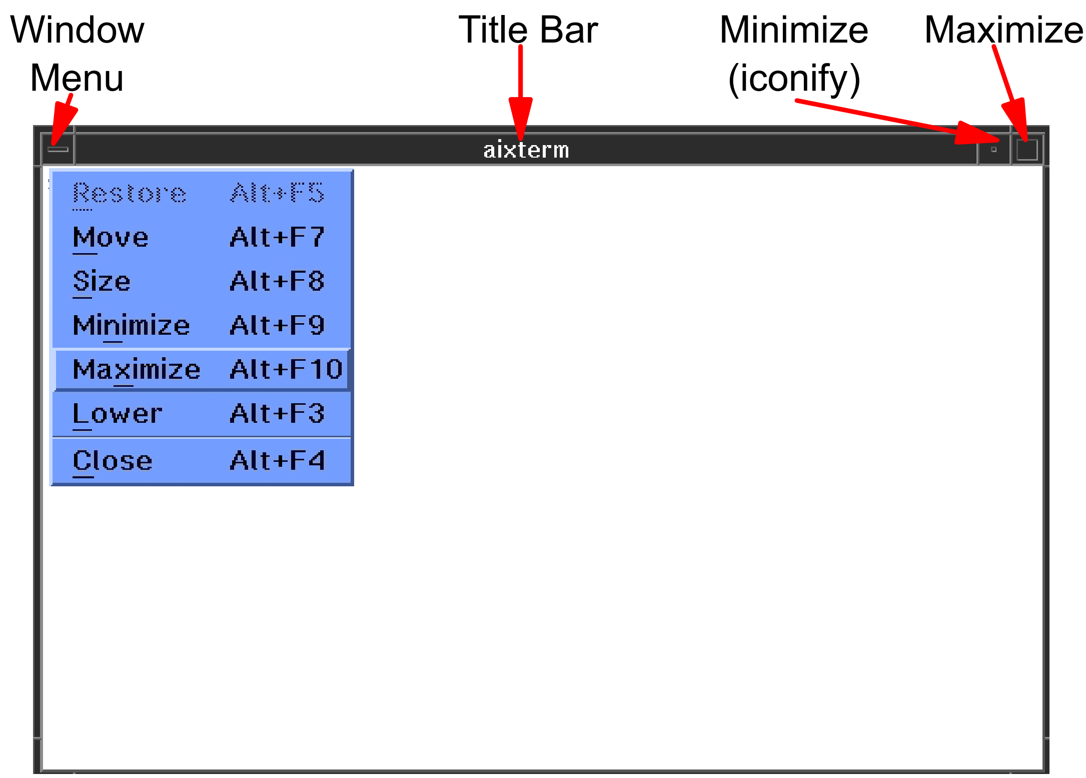


Figure 16-13. The Motif Window Frame AU139.0

Notes:

Introduction

Motif adds a *frame* around the windows. (Strictly speaking, the *window* is what is inside the frame, but many people speak of the frame and its contents as the window.) Many different AIXwindows applications can be started, but all can be run within windows that look and handle the same.

The frame allows the user to manipulate the window without affecting the program running in it. Each action requires that the mouse pointer is in the proper place.

Frame control

The *Title Bar* at the top of the window contains the name of the window. To *move* the window, place the mouse pointer on the title bar, press the left mouse button and drag the mouse.

The eight segments surrounding the window can be used to control the *size* of the window. Place the mouse pointer on one of these segments, click and hold the left mouse button and drag the mouse. This allows the window to be made longer, wider, narrower, shorter, and so forth.

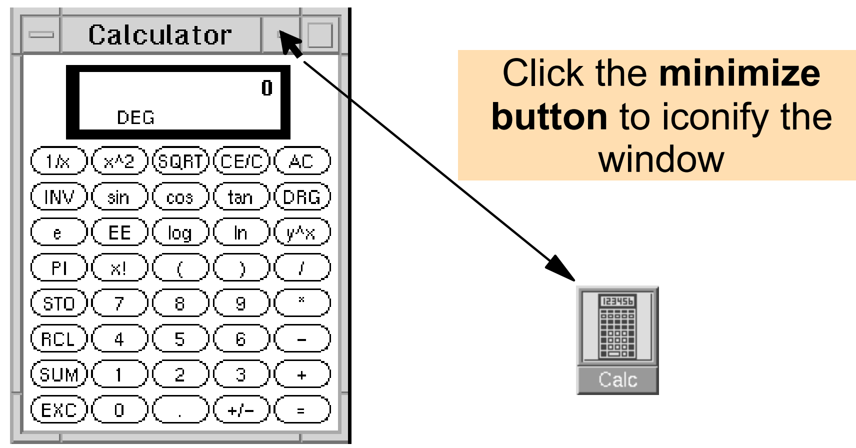
Click the large square in the upper right of the frame to *maximize* the window. The window will then expand to fill the entire screen.

Click the smaller square to the left of the *maximize button* to *minimize* or *iconify* the window.

The rectangle in the upper left of the frame can be used for two things. Use the left mouse button and double-click here to *close* the window. Single-clicking this area will display Motif's *window menu*. The *window menu* can also be used to move, resize, minimize, maximize, shuffle or close the window.

Icons

Icon = small graphics representation of a window



Normalized Window **Iconified Window**

Double clicking on the icon will restore the window

Figure 16-14. Icons AU139.0

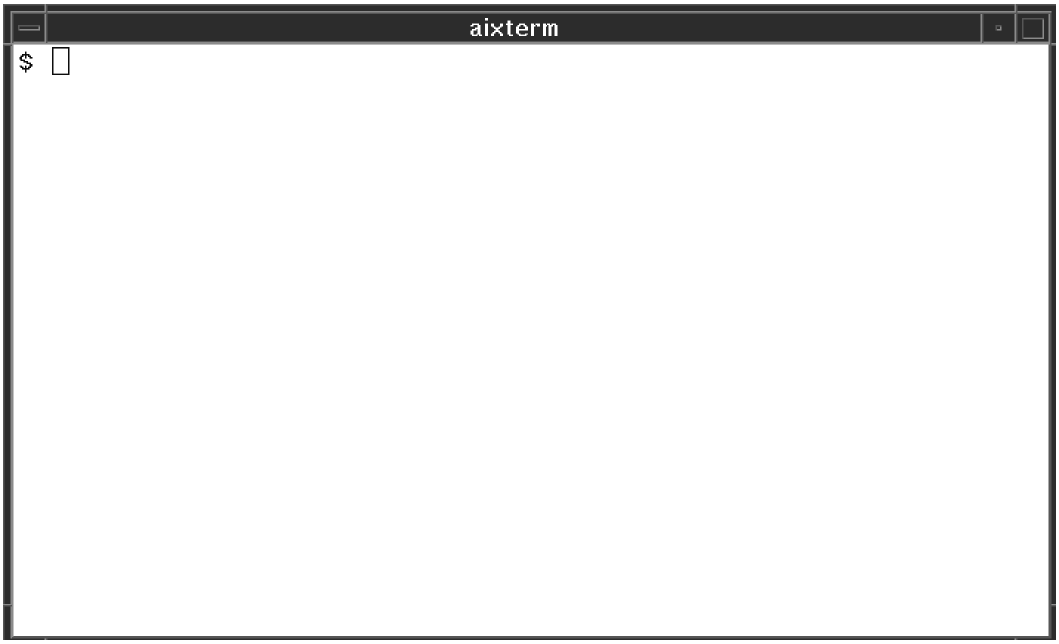
Notes:

Icons in AIXwindows

Icons are a useful way of managing screen space. By iconifying windows, the amount of clutter on the screen is reduced. Programs in an iconified window will continue to run.

A Motif icon image is a bitmap.

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**

Figure 16-15. The `aixterm` Window AU139.0

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-hand 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

aixterm Command Line Options

The following examples illustrate some of the most commonly used **command line options**:

```
aixterm -display SYSTEMB:0 &

aixterm -bg red -fg white -fn rom10 -geometry 80x40+0+0 &

aixterm -T PAYROLL &

aixterm -sb -leftscroll &
```


Figure 16-16. aixterm Command Line Options AU139.0

Notes:

There are many command line options for the **aixterm** command. Some options take on the opposite value if the - (minus sign) is changed to a + (plus sign).

Some of the commonly used aixterm command line options are:

-display <Name:Number>	Identifies the hostname and the X Server display number where the **aixterm** command is to run. If this is not specified, aixterm gets the host name and display number from the DISPLAY environment variable.
-bg <Color>	Specifies the color for the window background.
-fg <Color>	Specifies the color for the window foreground.
-fn <Font>	Specifies the normal sized text fontset.
-geometry <Geometry>	Specifies the location and dimensions of a window. The default is 80x25+0+0.
-T <Title>	Sets the title bar name but not the icon name.

The root Window

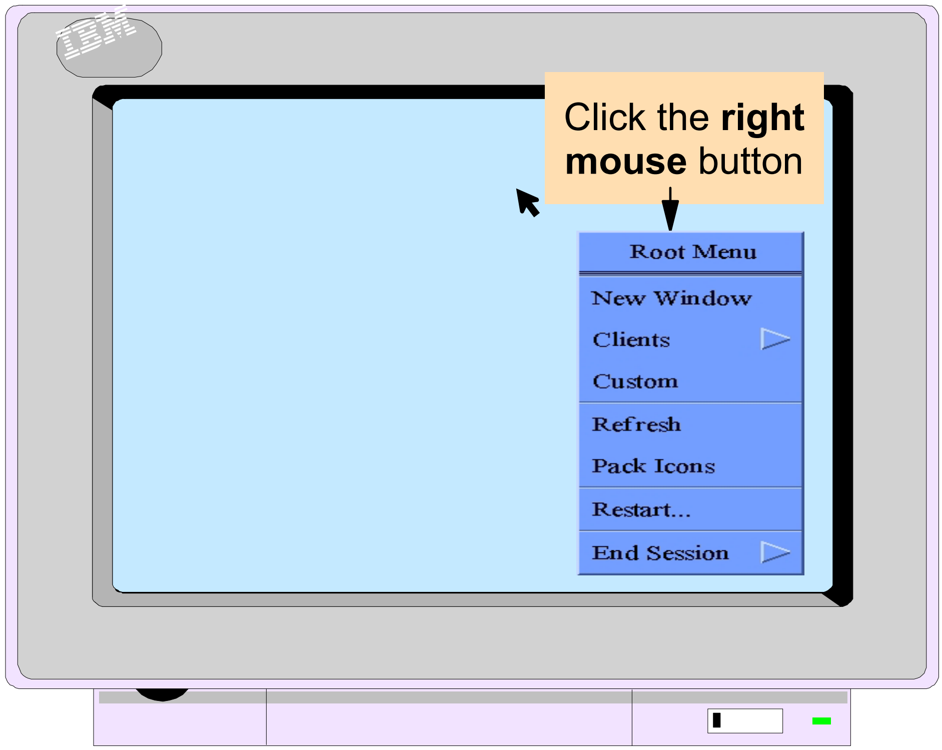


Figure 16-17. The root Window AU139.0

Notes:

root window options

Move the mouse pointer to the root window and press the right mouse button to display the `Root Menu`. Several options will be displayed:

`New Window`	Click the left mouse button here to create additional **`aixterm`** windows.
`Client`	Click the left mouse button to display an additional menu. This new menu will allow you to start another clock, SMIT, or lock the screen.
`Custom`	This item provides the ability to tailor the AIXwindows environment. Colors, fonts, focus policy, and so forth, can be tailored. More information on this option will be covered in the next unit.
`Refresh`	Redraw the display screen if system messages are overlaying its contents.
`Pack Icons`	Optimizes the icon layout on the screen if using an icon box.

`Restart`	Stops and restarts the **`mwm`**.
`End Session`	Stops the **`mwm`** and ends AIXwindows. The key sequence **`<Ctrl><Alt><Backspace>`** can also be used to end the AIXwindows session.

root configuration file

The root menu can be customized using the **$HOME/.mwmrc** file. This file will be discussed in the next unit.

Running a Client on Another System

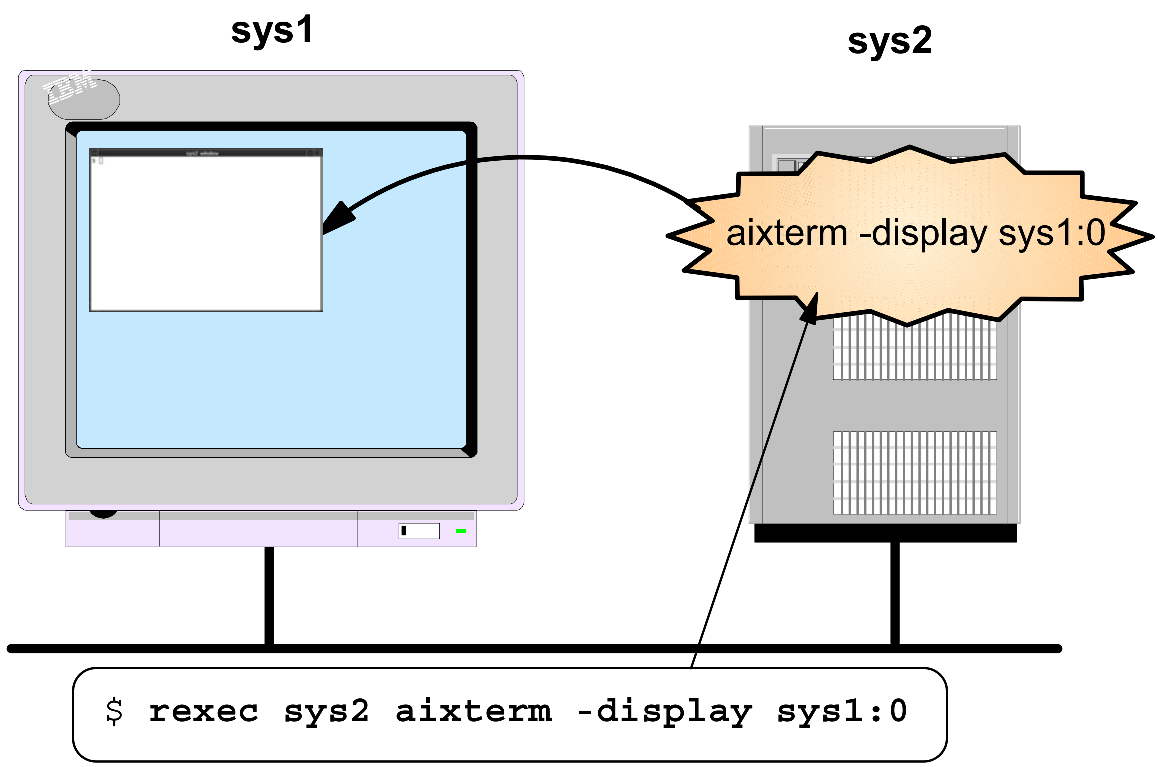


Figure 16-18. Running a Client on Another System AU139.0

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 (i.e. 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.

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

Figure 16-19. The xhost Command AU139.0

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.

Checkpoint

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?

Figure 16-20. Checkpoint AU139.0

Notes:

Exercise: Using AIXwindows

Figure 16-21. Exercise: Using AIXwindows AU139.0

Notes:

After completing the exercise, you will be able to:

- Start AIXwindows.
- Manipulate screen windows using AIXwindows.
- Open a new **aixterm** window.

An optional exercise part shows how you can use AIXwindows in a client/server environment.

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 AIXwindows to move, resize, maximize, minimize and close windows.
- Use the `DISPLAY` variable or the **`-display`** option to designate which server a client will send its output to.

Figure 16-22. Unit Summary AU139.0

Notes:

Unit 17. Customizing AIXwindows

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

Figure 17-1. Unit Objectives AU139.0

Notes:

AIXwindows Startup Overview

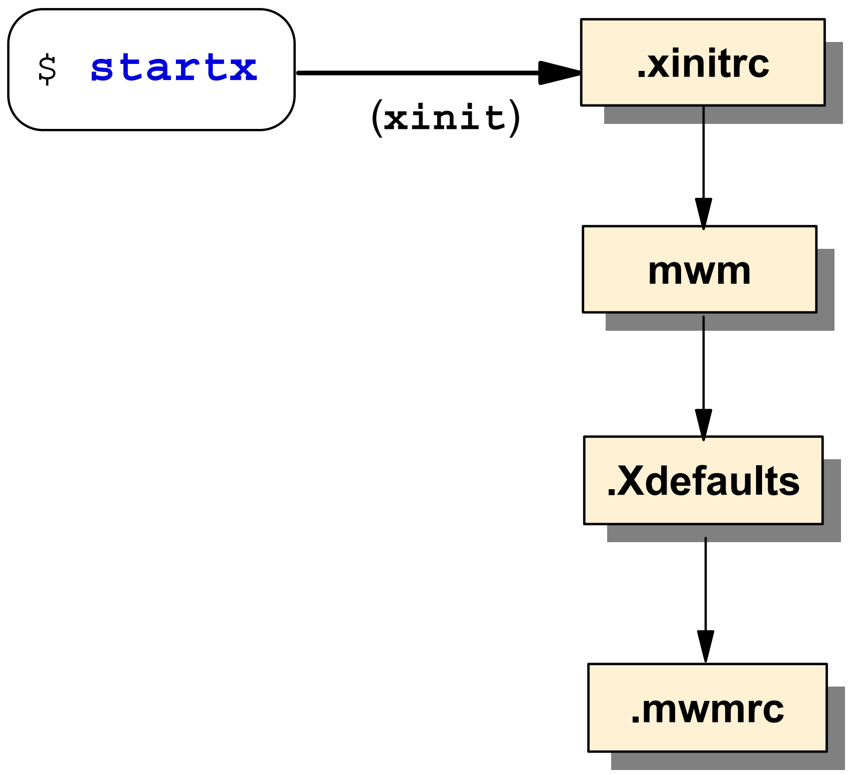


Figure 17-2. AIXwindows Startup Overview AU139.0

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 RS/6000 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.

.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
...

#***********************************************************
#
#   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).
#
#***********************************************************
xsetroot -solid grey60
aixterm -geometry 80x25+0-0 &
exec mwm
```

xclock

Background color

aixterm

Motif Window Manager

Figure 17-3. **.xinitrc** AU139.0

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.

Geometry Specifications for Clients

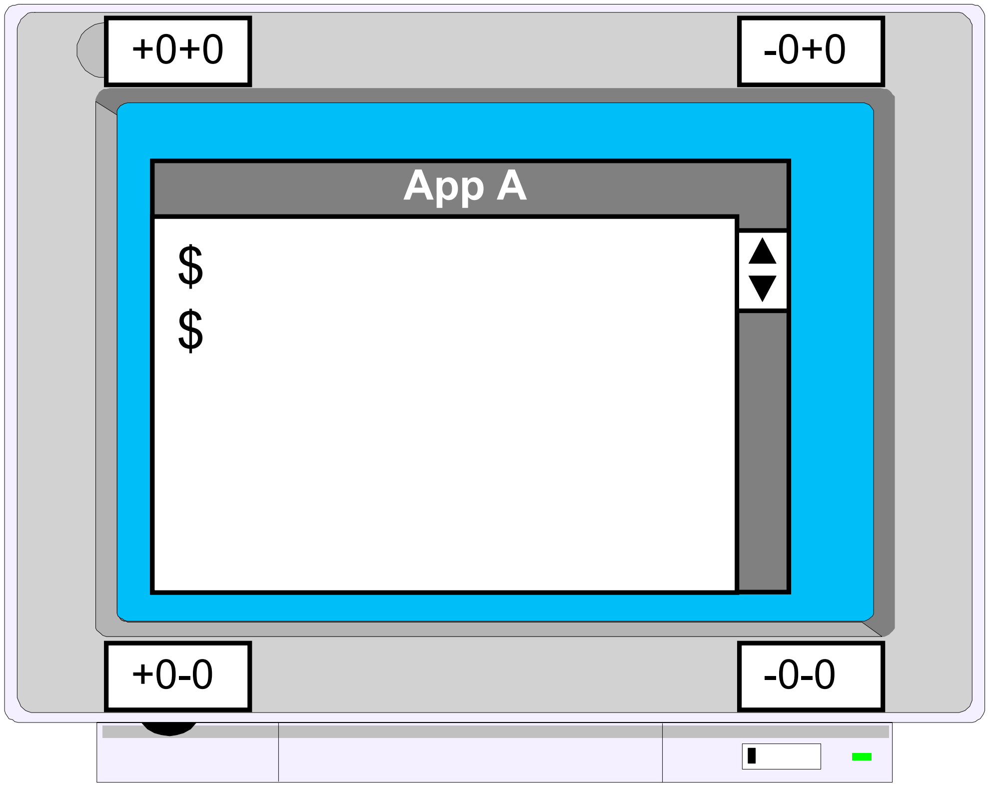


Figure 17-4. Geometry Specifications for Clients AU139.0

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 hand side of the screen. A negative value means an offset from the right hand 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 hand edge of the screen and 300 pixels from the top of the screen relative to the left hand corner of the client.

Similarly, the **xclock** will appear in the top right hand corner of the screen and will be 200 pixels in width and 250 pixels in height.

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
```


Figure 17-5. The Color Database AU139.0

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.

Fonts

- Fonts are stored in the directory **/usr/lib/X11/fonts**
- To **list all the fonts** available use the command:
  `$ custom -e font`

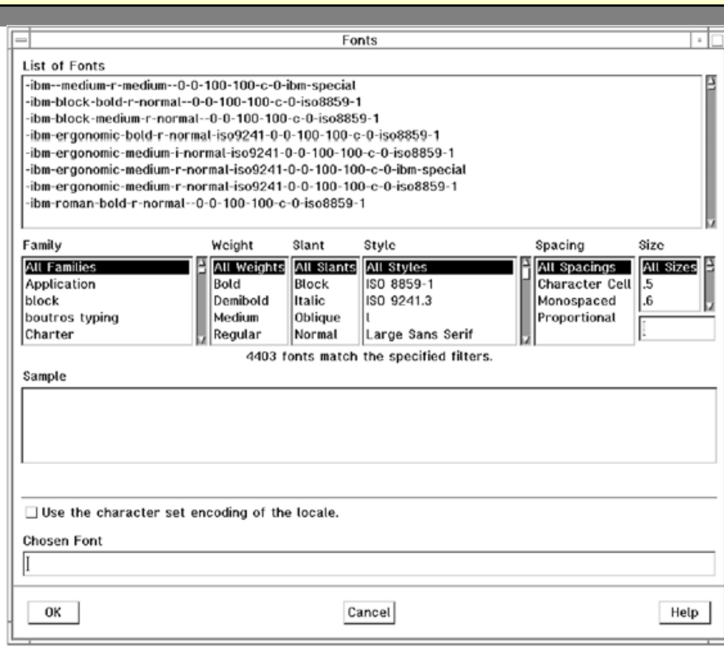


Figure 17-6. Fonts AU139.0

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:

- foundry
- font family
- weight
- slant
- set width
- additional style
- pixels
- points
- horizontal resolution
- vertical resolution
- spacing
- average width
- character set

.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
```


Figure 17-7. .Xdefaults AU139.0

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.**

.mwmrc

```
$ cp /usr/lpp/X11/defaults/Motif1.2/system.mwmrc $HOME/.mwmrc
$ vi $HOME/.mwmrc
```

Do not directly edit the system wide system.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
```


Figure 17-8. **.mwmrc** AU139.0

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.

Exercise: Customizing AIXwindows (1)

Figure 17-9. Exercise: Customizing AIXwindows (1) AU139.0

Notes:

After completing the exercise, you will be able to:

- Customize the **.xinitrc** file.
- Customize the **.Xdefaults** file.

AIXwindows Custom Application

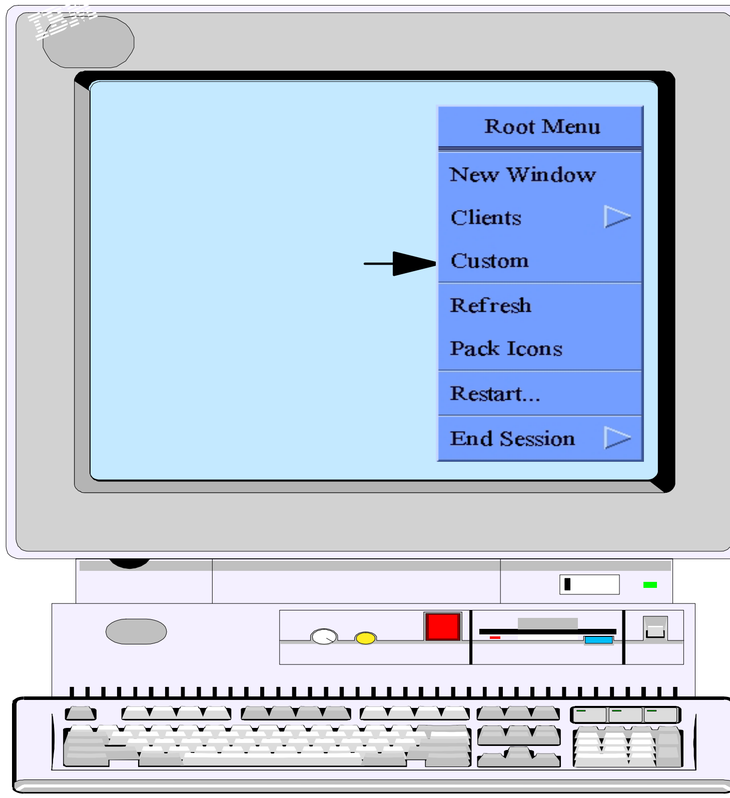


Figure 17-10. AIXwindows Custom Application AU139.0

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.

The Custom Window

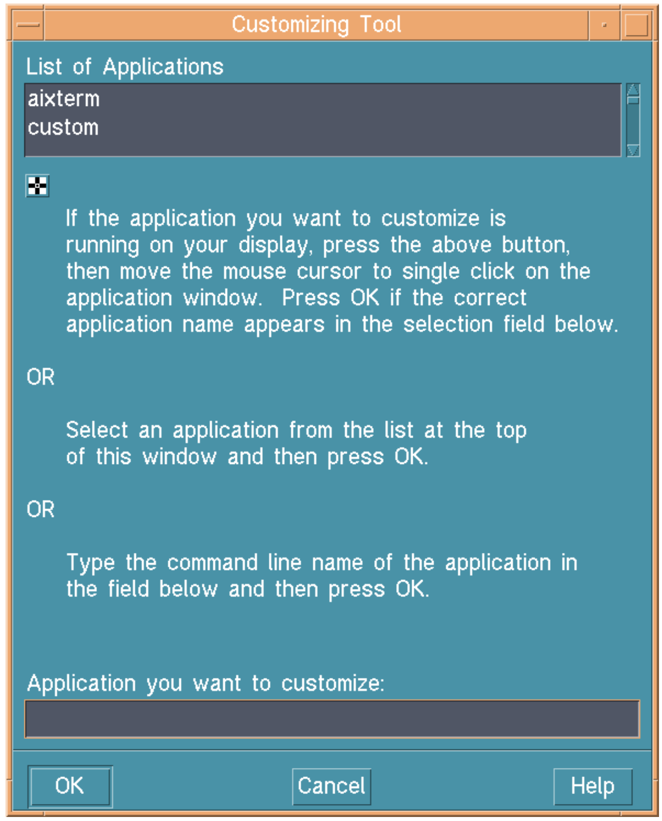


Figure 17-11. The Custom Window AU139.0

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 scroll bar 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 it's 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**.

Customizing an `aixterm`

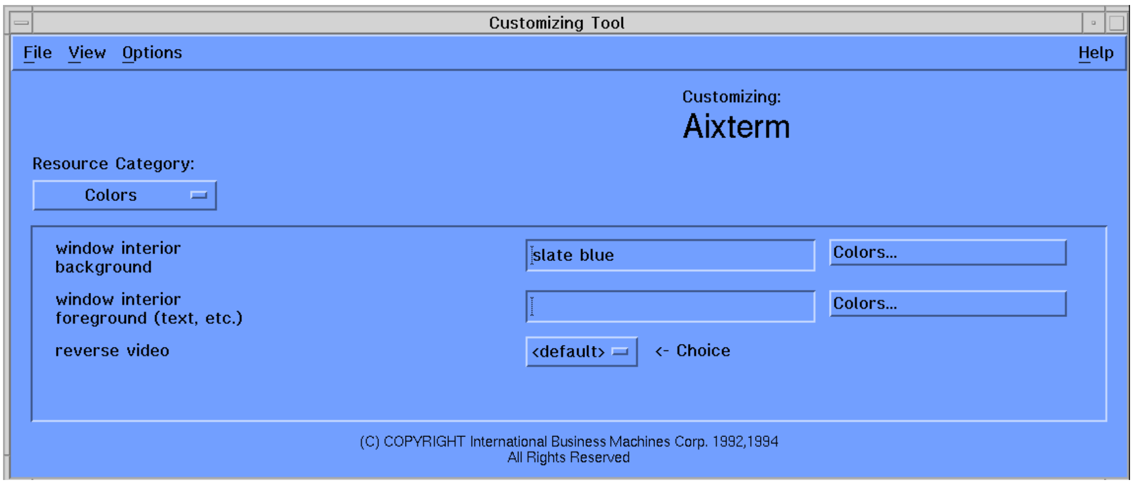


Figure 17-12. Customizing an **`aixterm`** AU139.0

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), scroll bar 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
```

The AIXwindows Color Browser

Figure 17-13. The AIXwindows Color Browser AU139.0

Notes:

Color browser

Above is the AIXwindows color browser. In the top window is a list of colors. A scroll bar 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.

Saving the Customized Changes

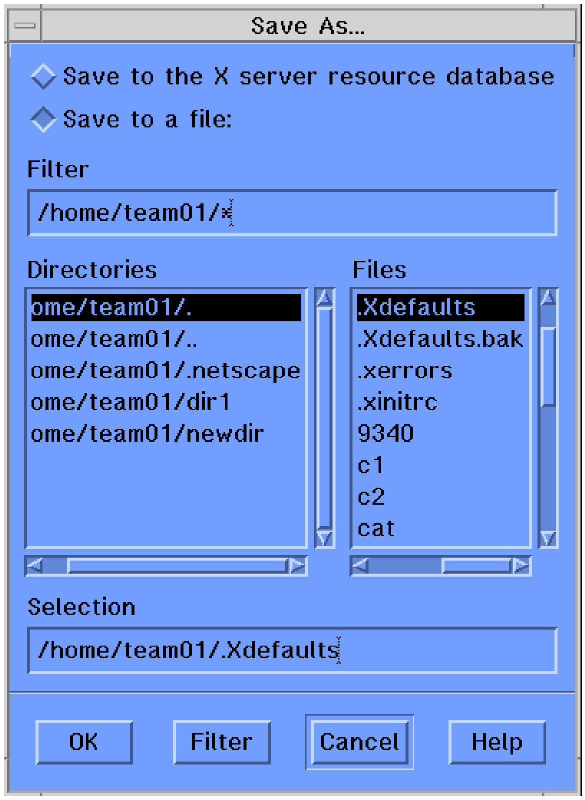


Figure 17-14. Saving the Customized Changes AU139.0

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.

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**

Figure 17-15. The xsetroot Command AU139.0

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 *DISPLAY* 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 **custom -e picture** command.

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?

Figure 17-16. Checkpoint AU139.0

Notes:

Exercise: Customizing AIXwindows (2)

Figure 17-17. Exercise: Customizing AIXwindows (2) AU139.0

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*.

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.

Figure 17-18. Unit Summary AU139.0

Notes:

Unit 18. Using the Common Desktop Environment (CDE)

What This Unit Is About

This unit provides an introduction to the functions of the Common Desktop Environment (CDE).

What You Should Be Able to Do

After completing this unit, you should be able to:

- Describe the goal of the CDE environment
- Use the various CDE components

How You Will Check Your Progress

Accountability:

- 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:

- Describe the goal of the Common Desktop Environment (CDE)
- Use the various CDE components

Figure 18-1. Unit Objectives AU139.0

Notes:

Common Desktop Environment (CDE)

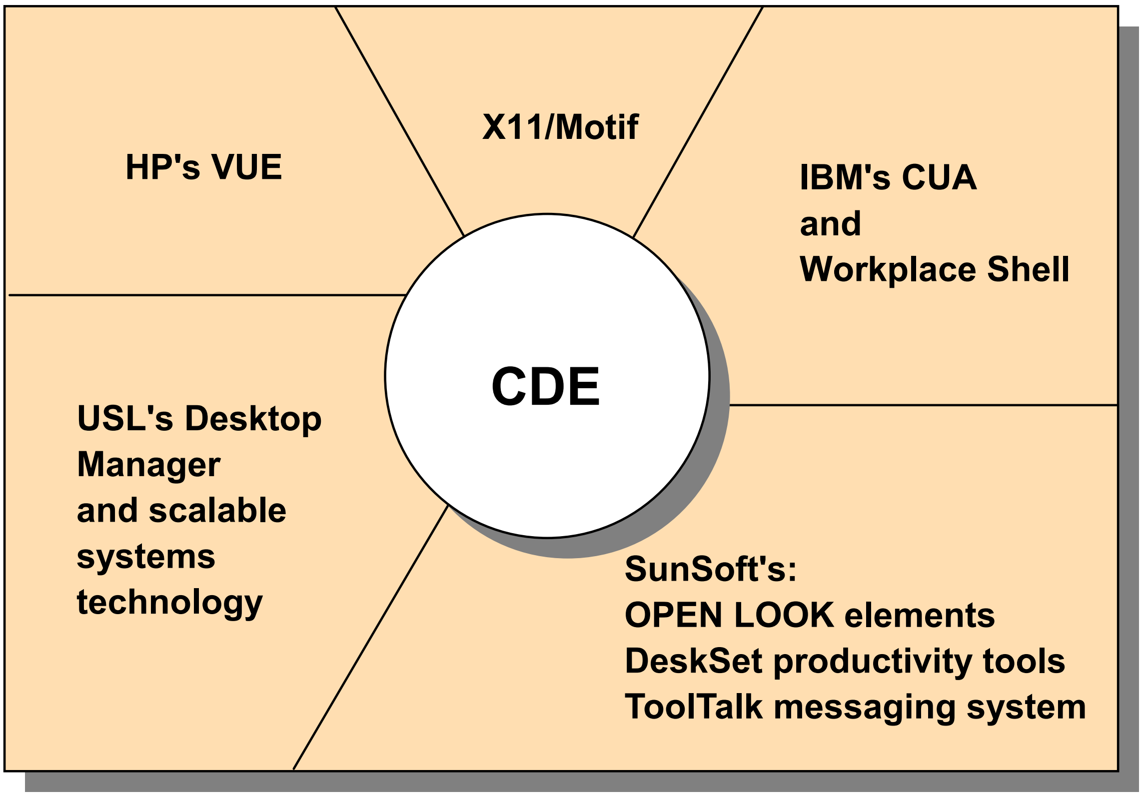


Figure 18-2. Common Desktop Environment (CDE) AU139.0

Notes:

Introduction

The Common Desktop Environment (CDE) provides a common user interface for the UNIX environment.

Where did CDE come from?

Even though X Windows provided a common base and many UNIX system 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 Window.

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 is the desktop interface included in all releases of AIX 5L.

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
- The **Session** Manager

Figure 18-3. The Components of the CDE Desktop AU139.0

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
- The Session Manager: Used to maintain desktop look between sessions

These components will be covered in this and the next unit.

The Login Manager

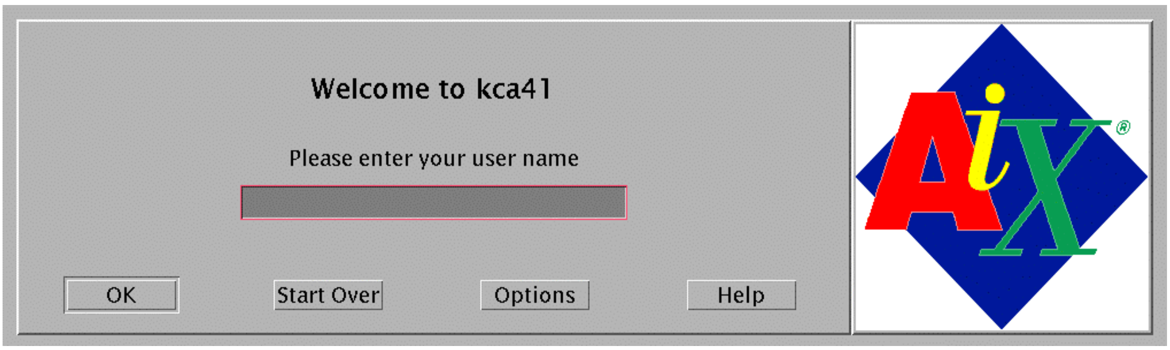


Figure 18-4. The Login Manager AU139.0

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**.

$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
```


Figure 18-5. **$HOME/.dtprofile** AU139.0

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.

Front Panel

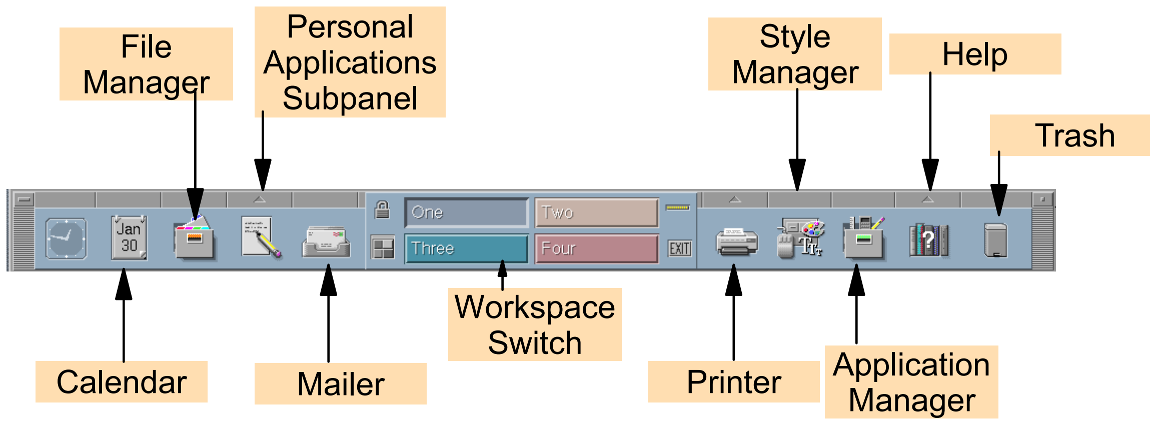


Figure 18-6. The Front Panel AU139.0

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.

Front Panel - Subpanels

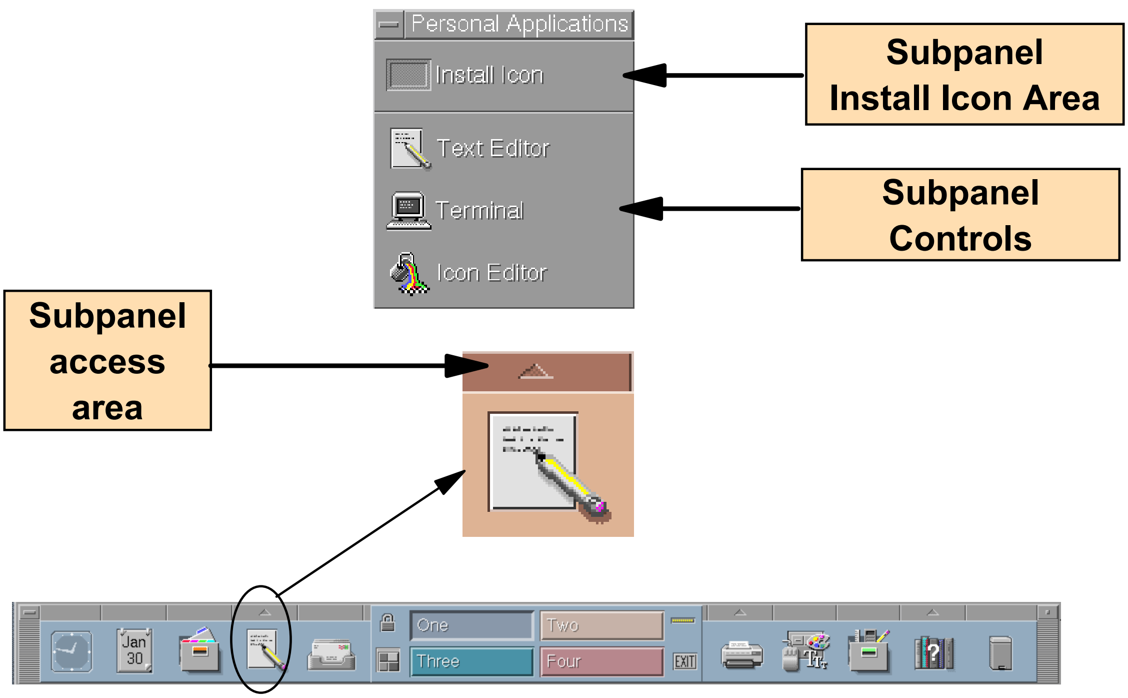


Figure 18-7. Front Panel - Subpanels AU139.0

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 labelled 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.

Front Panel - Further Controls

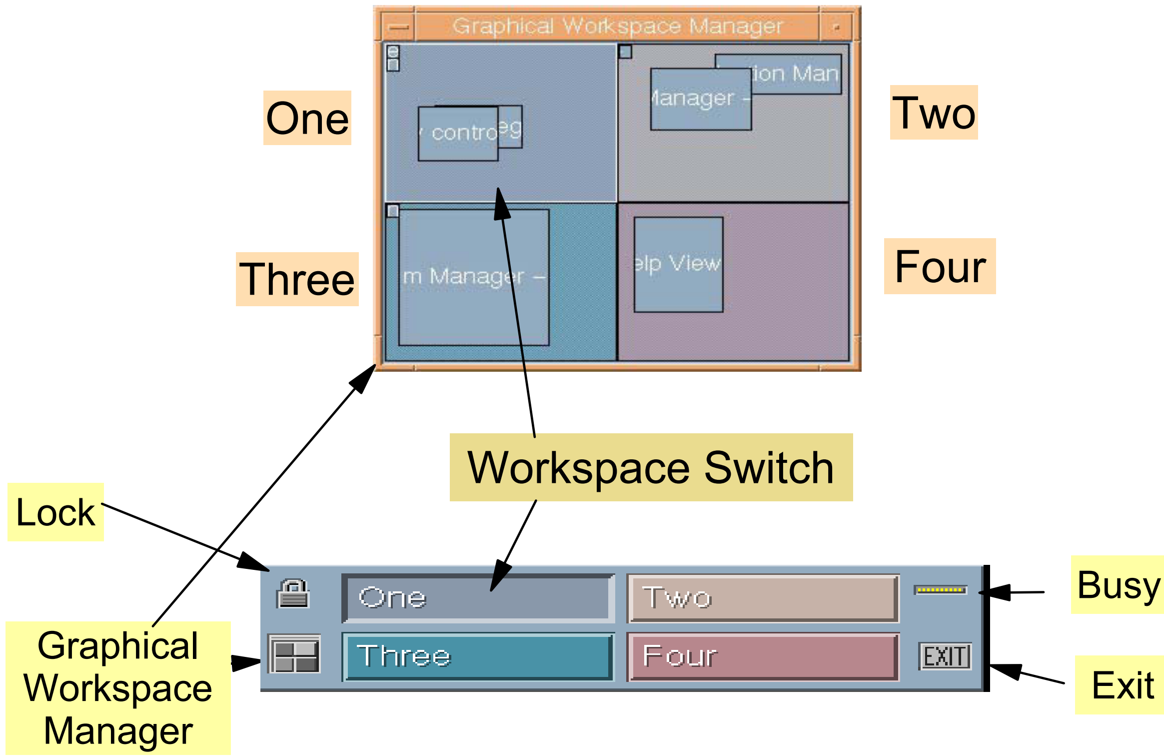


Figure 18-8. Front Panel - Further Controls AU139.0

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.

Activity: What's This?

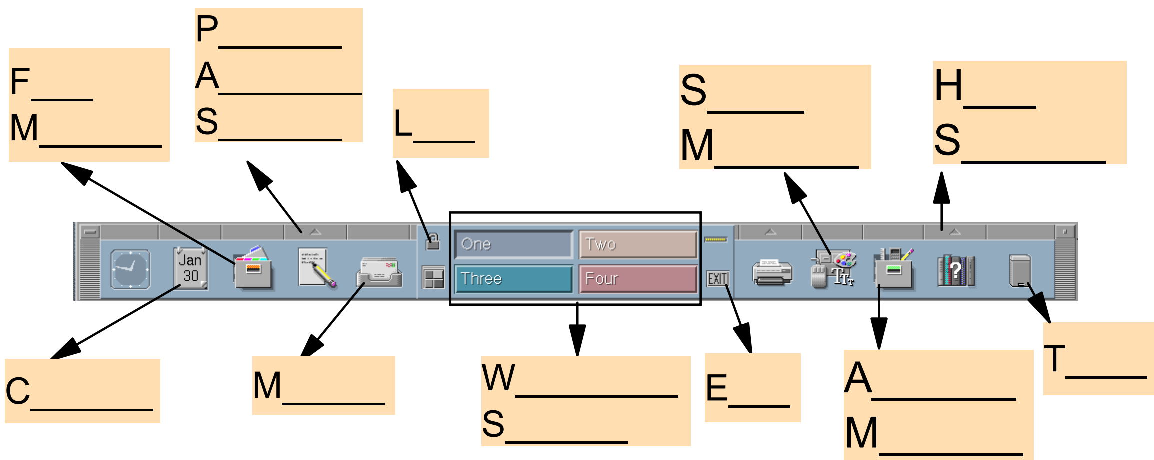


Figure 18-9. Activity: What's This? AU139.0

Notes:

Activity

This activity reviews the most important CDE components. Take some time and complete the visual.

The Style Manager

The **Style Manager** allows interactive **customization** of the desktop environment

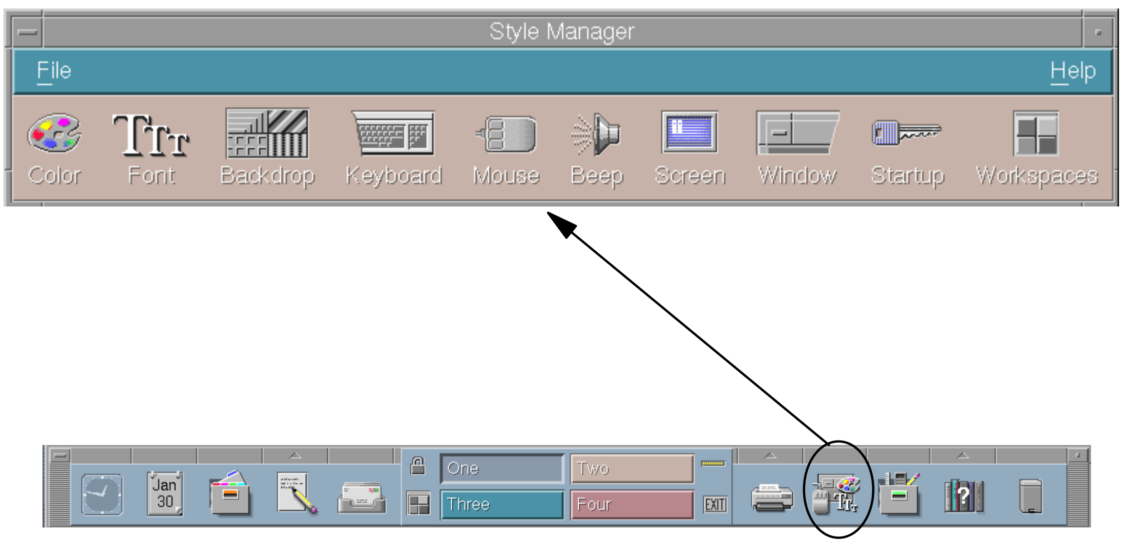


Figure 18-10. The Style Manager AU139.0

Notes:

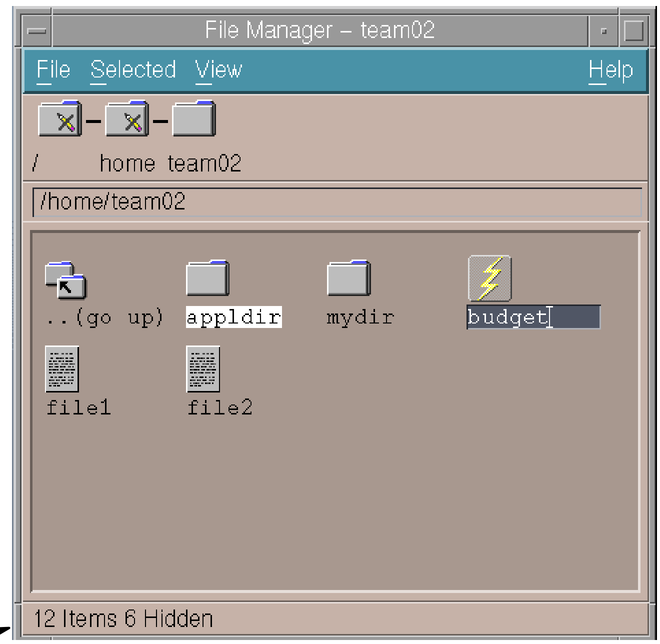
Customizing the workspace

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.

The File Manager

- Navigate file system
- Work with files and directories
- Drag and drop actions
- Launch applications



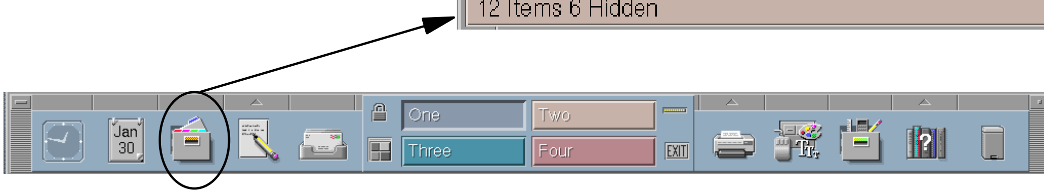


Figure 18-11. The File Manager AU139.0

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.

The Application Manager

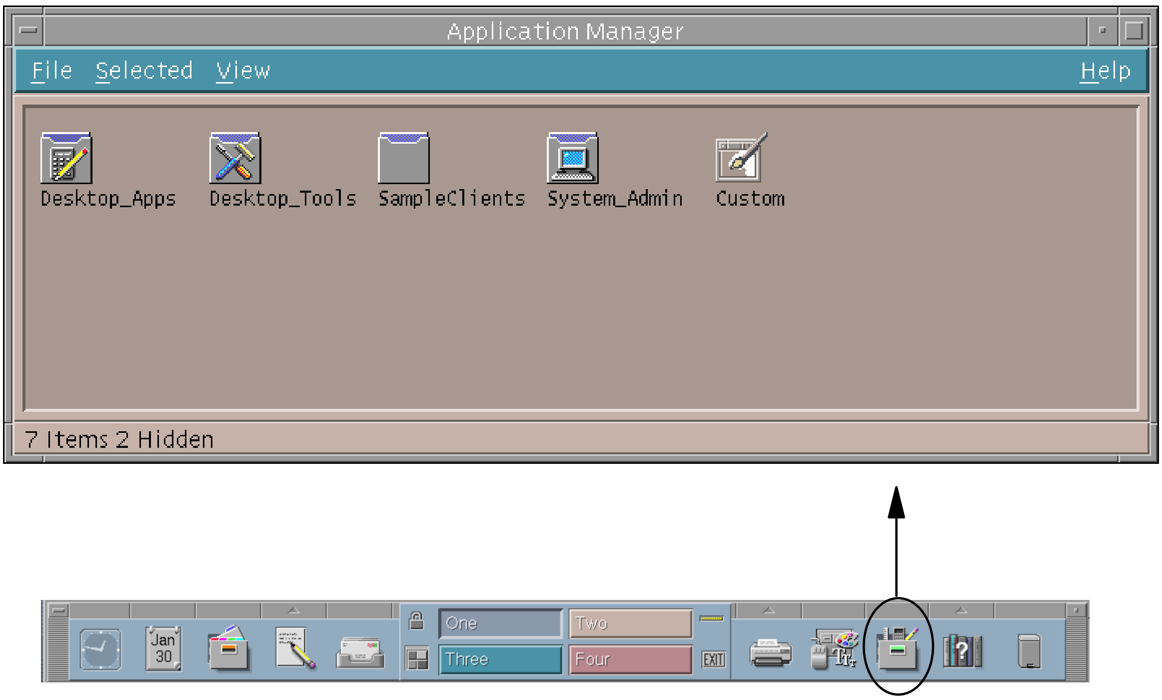


Figure 18-12. The Application Manager AU139.0

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.

The Personal Applications Manager

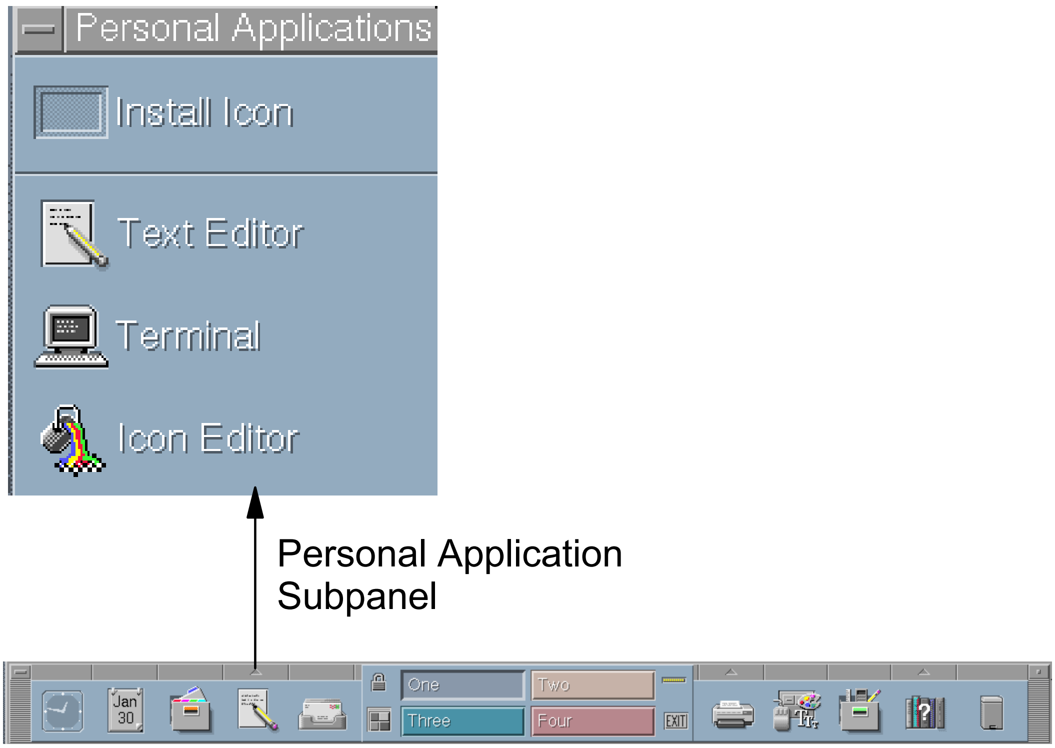


Figure 18-13. The Personal Applications Manager AU139.0

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`**.

The Terminal Emulator

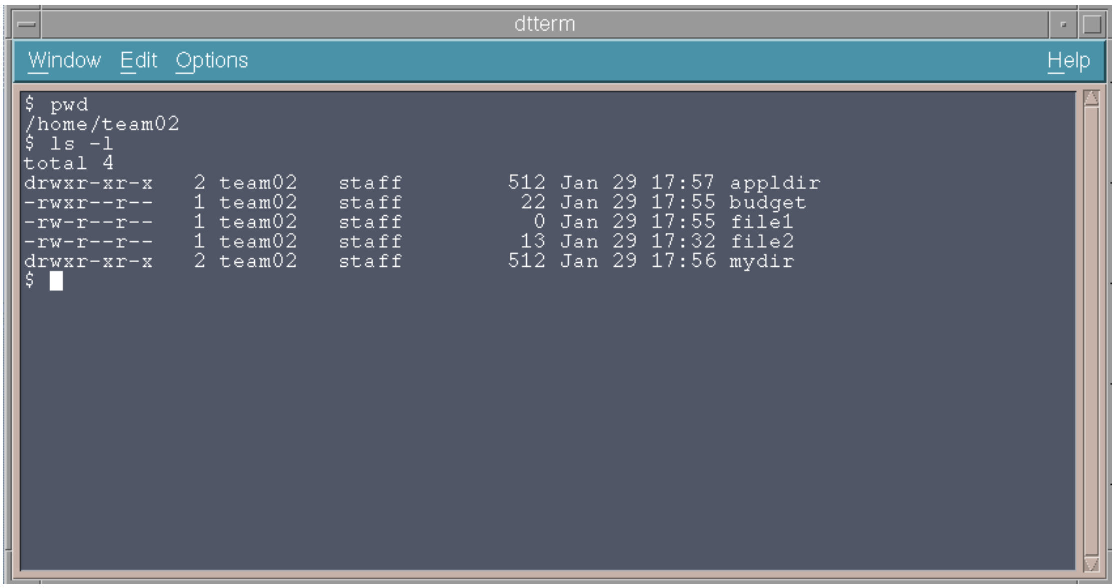


Figure 18-14. The Terminal Emulator AU139.0

Notes:

`dtterm`

The Desktop Terminal Emulator **`dtterm`** can be used in place of the **`aixterm`** to enter AIX commands.

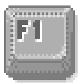
The **`dtterm`** contains a scroll bar as well as a menu bar.

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.

The Help System

- Readily available from:



Help Key

Application Help Menu

Help Subpanel

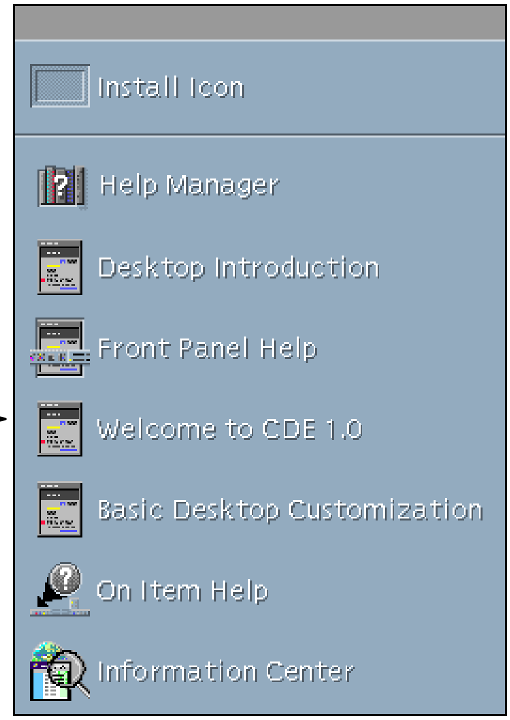


Figure 18-15. The Help System AU139.0

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
- Choose `Help` from any application's pull-down menu
- Click 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.

The Session Manager

The **Session Manager** is responsible for **setting up** the **user's desktop environment** after login processing has completed.

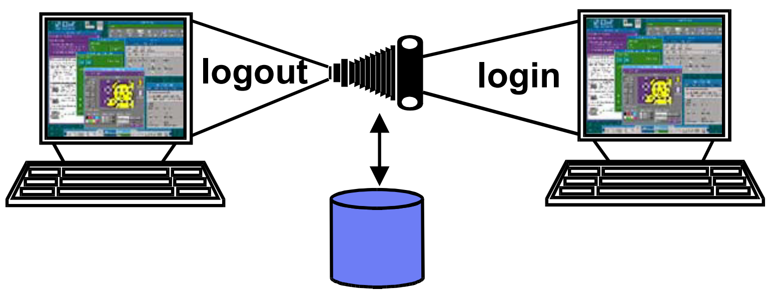


Figure 18-16. The Session Manager AU139.0

Notes:

Setting up the desktop environment

The CDE Session Manager is responsible for setting up the user's desktop environment after login processing has completed. The Session Manager can either restore the user's initial desktop configuration (the home session) or restore a snapshot of the running desktop saved by the session manager during the last logout (called the current session).

By default, users will log in to their current session. This option can be changed through the `Style Manager` by choosing **`Startup`**. A user can also choose to be asked which session to bring up at startup.

The CDE Mail Program

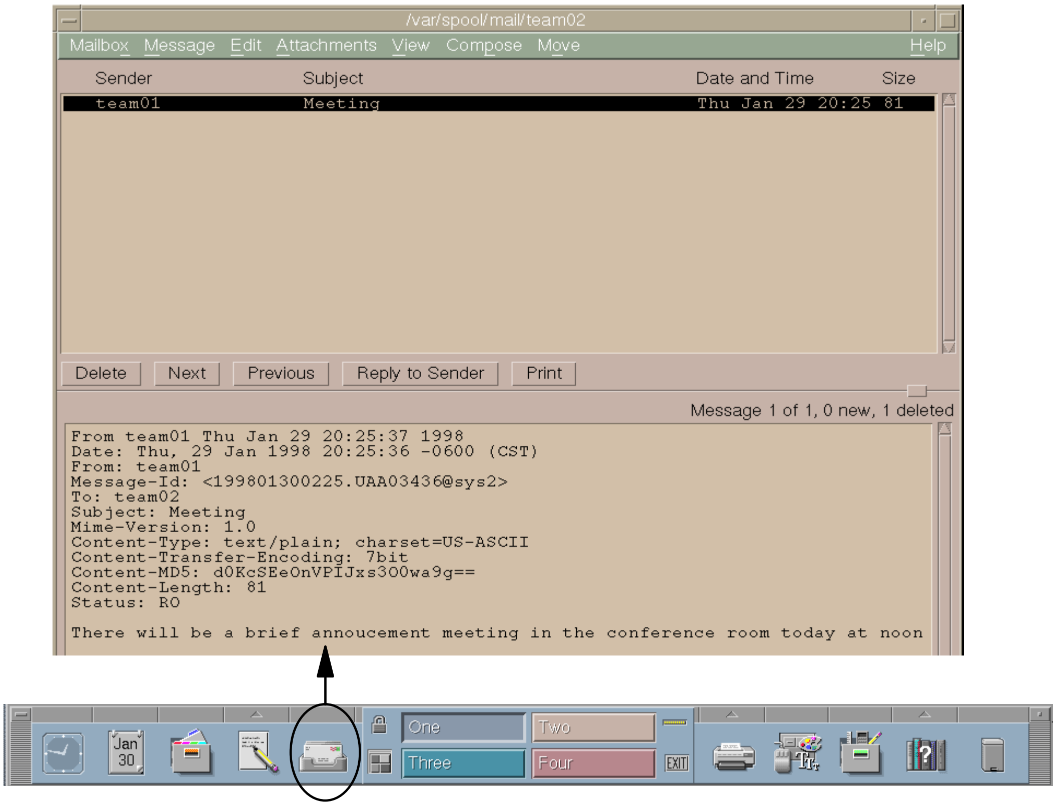


Figure 18-17. The CDE Mail Program AU139.0

Notes:

Accessing mail with CDE

The CDE Mail tool permits users to view, file, compose, send and receive electronic messages. CDE-Mail will provide electronic mail services to other clients on the CDE Desktop.

Supported tasks are performed via a window-based interface.

CDE Mail window

The Mail main window contains two working areas: One to display a list of messages and the other to display the contents of the currently selected message.

Viewing a mail message

To view the next message, simply select the next item in the top window and its contents will be shown in the `Message View` area below. The same can be done pressing the **`Next`** or **`Previous`** buttons.

Sending mail

To send mail to another user, click **`Compose`** in the menu bar and then choose **`New Message`**. If replying to mail, Compose provides options to forward the message, reply to the sender, and so forth.

Removing mail messages

To delete, click the **`delete`** button. To save the mail message to a file, click **`Move`**, then **`Other Mailboxes`**. It is then possible to save the mail to a filename that you specify.

CDE Mail Program - Send a Message

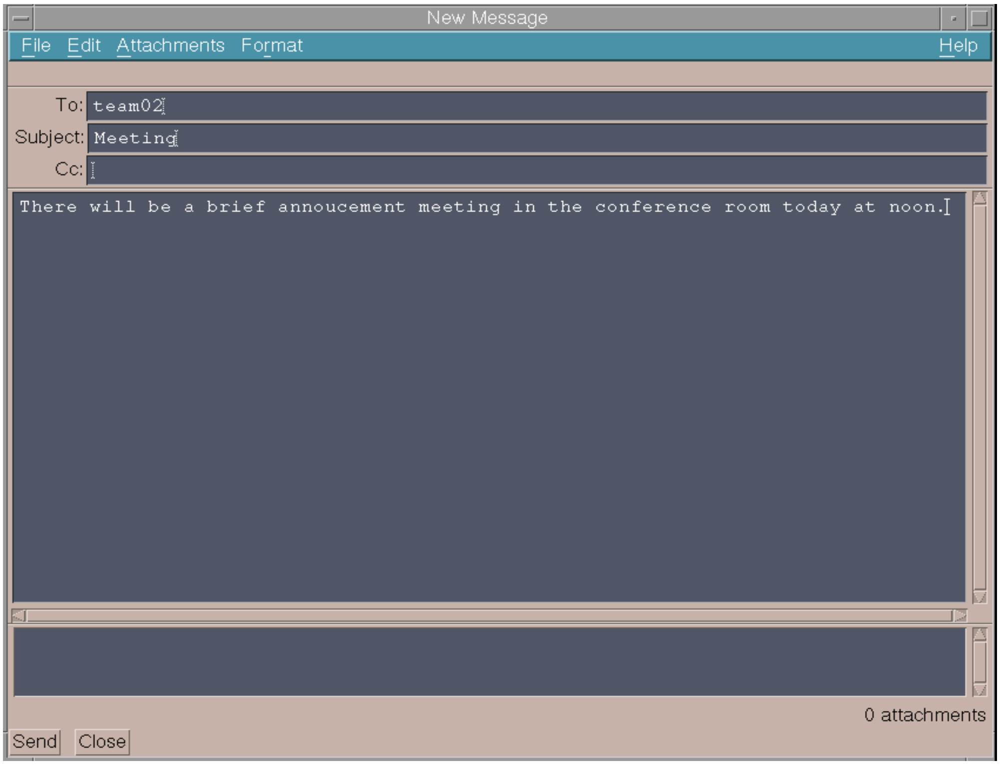


Figure 18-18. CDE Mail Program - Send a Message AU139.0

Notes:

Sending mail in CDE

From the `Compose` option on the main window menu bar, it is possible to send a note, forward a note or reply to a note. In either case, the same window is opened. The only difference is that when forwarding or replying, the fields `To` and `Subject` are already filled in.

The `Send Message` window contains:

- Menu bar
- `To`, `Subjects` and `cc` fields
- Scrollable Message Text Area
- `Send` button

Attaching files

From the `File` option it is possible to include a file into the `Message Text` area, save the message text to a file, print it and log the message. The `Send` function and the `Close` function return to the main window.

Editing mail

`Cut`, `Copy` and `Paste` functions which operate on the text, are available from the `Edit` option.

The Calendar Manager

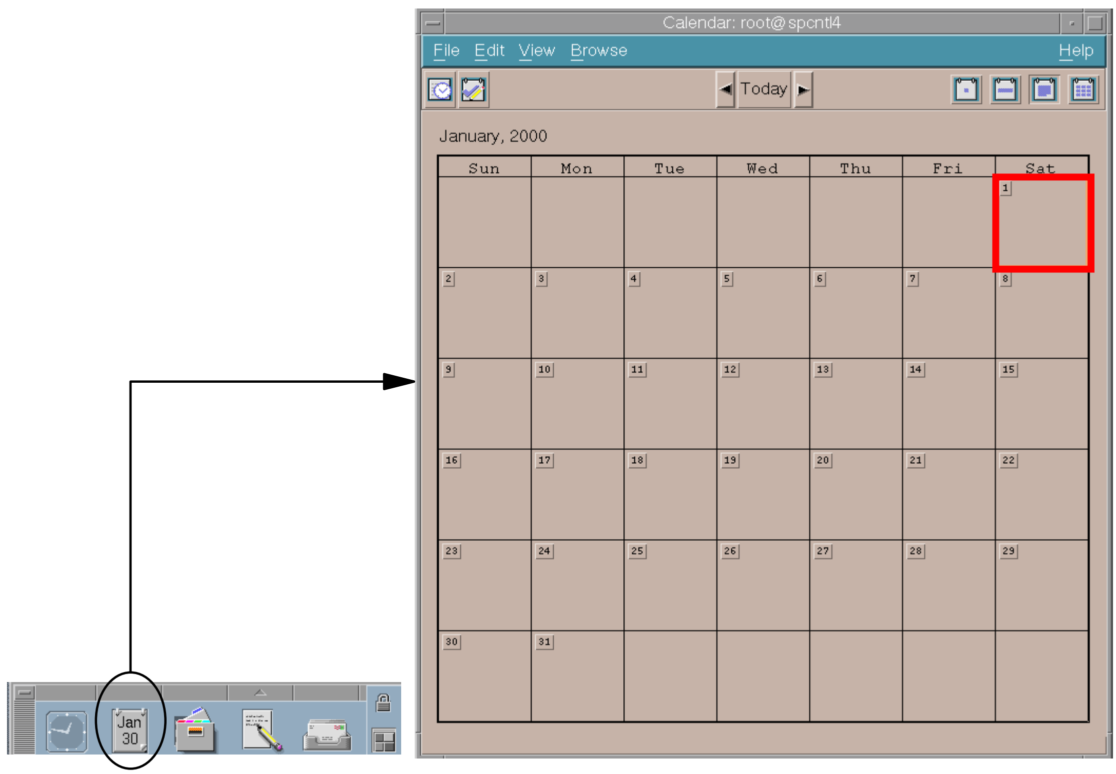


Figure 18-19. The Calendar Manager AU139.0

Notes:

Calendar functions in CDE

The `Calendar` is a desktop application that enables you to schedule appointments and To Do items, set reminders, make and print appointments and To Do lists, browse other calendars and schedule group appointments.

To start the `Calendar`, click the `Calendar control` on the front panel.

Calendar tools

The `Calendar` contains a Tool Bar, which provides a number of functions. Starting with the left, the various tool bars support the following functions:

- `Appointment Editor`: Add, update or delete appointments.
- `To Do Editor`: Create and modify reminders for yourself.

- `<Today>`: The left arrow shows the previous view. Our example shows the January 2000 calendar. Pressing the left arrow would display December 1999. Similarly, pressing the right arrow would display February 2000. Clicking Today will highlight the current day.
- `Day View`: Changes the view to show an hourly schedule for the day. The current, previous and next months calendars are also displayed.
- `Week View`: Shows a weekly schedule.
- `Month View`: The default view, which displays the entire month.
- `Year View`: Displays the calendar for the entire year. Appointments are not displayed.

Calendar Appointments

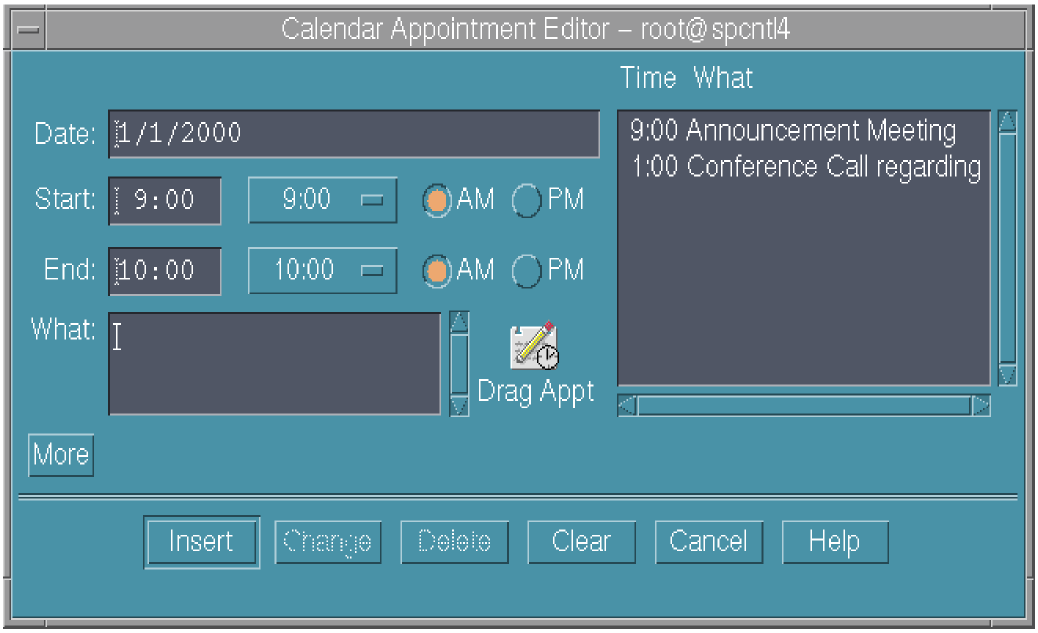


Figure 18-20. Calendar Appointments AU139.0

Notes:

Scheduling appointments in the calendar

Use the **`Appointment Editor`** to insert, change or delete appointments on your calendar. The Appointment Editor is accessible from the Calendar Tool Bar or from the **`Edit`** option on the calendar menu bar.

Choose the date and the starting and ending times for an appointment. Then, type in the details of the appointment in the `What` box. To record the appointment, click **`Insert`**. The `Time What` box will display a list of all appointments for that day.

To change a specific appointment, click that appointment in the `Time What` box. The appointment can then be edited in the `What` box.

Click **`More`** to see an extended window where reminders can be set for any appointment. Here it is also possible to set the privacy level of an appointment.

Checkpoint

1. True or false? CDE is designed as a common user interface for the UNIX environment.
2. 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

3. True or false? Any environment variables set in .profile will be used by default in the CDE environment.

Figure 18-21. Checkpoint AU139.0

Notes:

Exercise: Using the CDE

Figure 18-22. Exercise: Using the CDE AU139.0

Notes:

After completing the exercise, you will be able to:

- Recognize the various CDE controls on the front panel.
- Use the *Help Manager*.
- Start a *Terminal* window.
- Use the *File Manager*.

Optionally, use the *Calendar* control to view the calendar, set appointments and create reminders.

Unit Summary

- The CDE provides a common user interface for the user environment.
- After logging in to CDE, the Front Panel provides access to the various CDE controls:
  - Calendar
  - File Manager
  - Personal Applications
  - Mail
  - Printing Functions
  - The Style Manager
  - The Application Manager
  - The Help Manager
  - Trash
- The Front Panel also contains a Workspace Switch and the ability to lock a display or log out.

Figure 18-23. Unit Summary AU139.0

Notes:

Unit 19. CDE User Customization

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

Figure 19-1. Unit Objectives AU139.0

Notes:

Customizing CDE

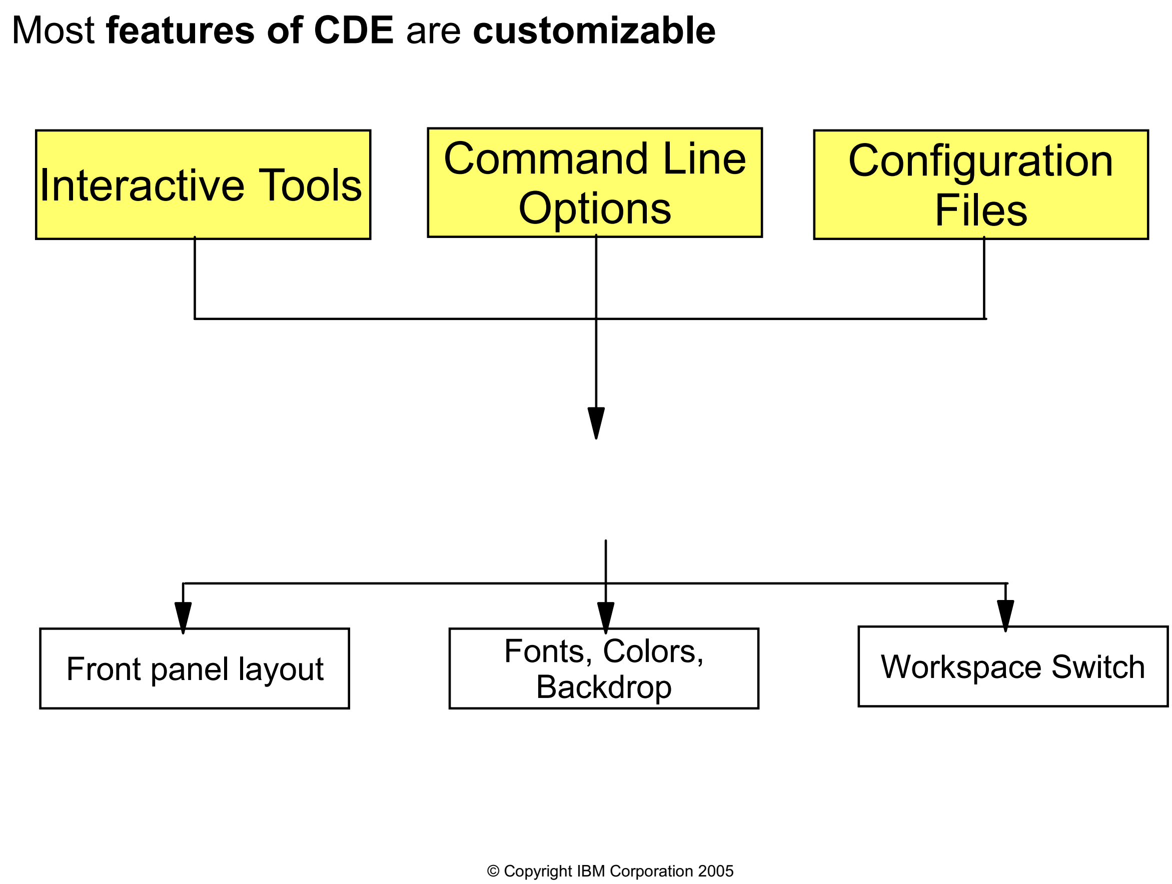


Figure 19-2. Customizing CDE AU139.0

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.

Style Manager Overview

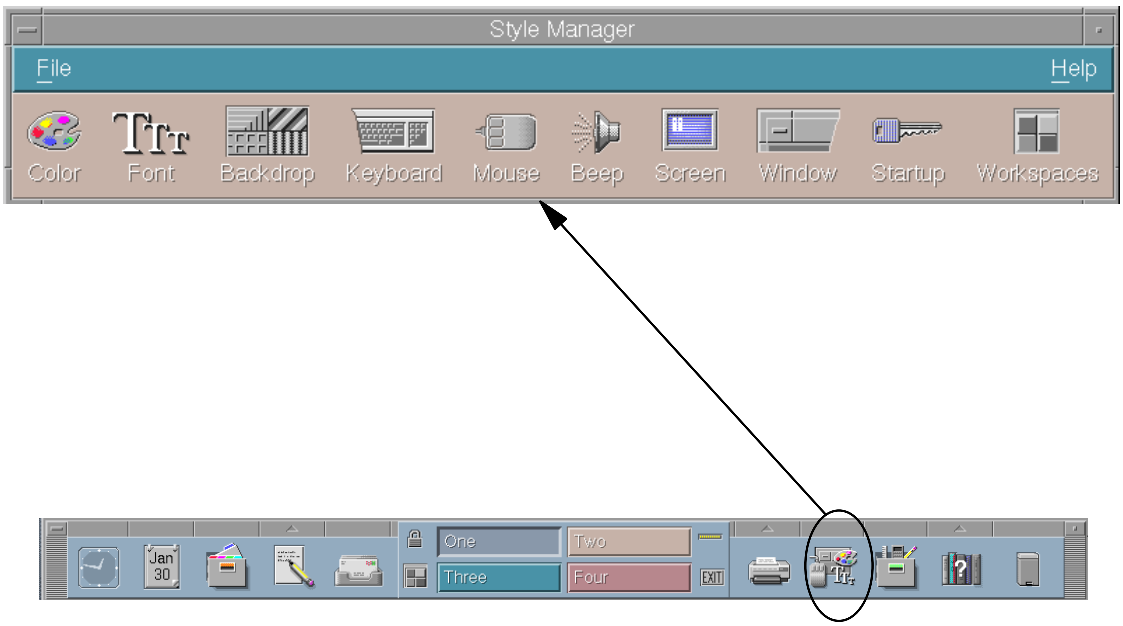


Figure 19-3. Style Manager Overview AU139.0

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)

Style Manager - Colors

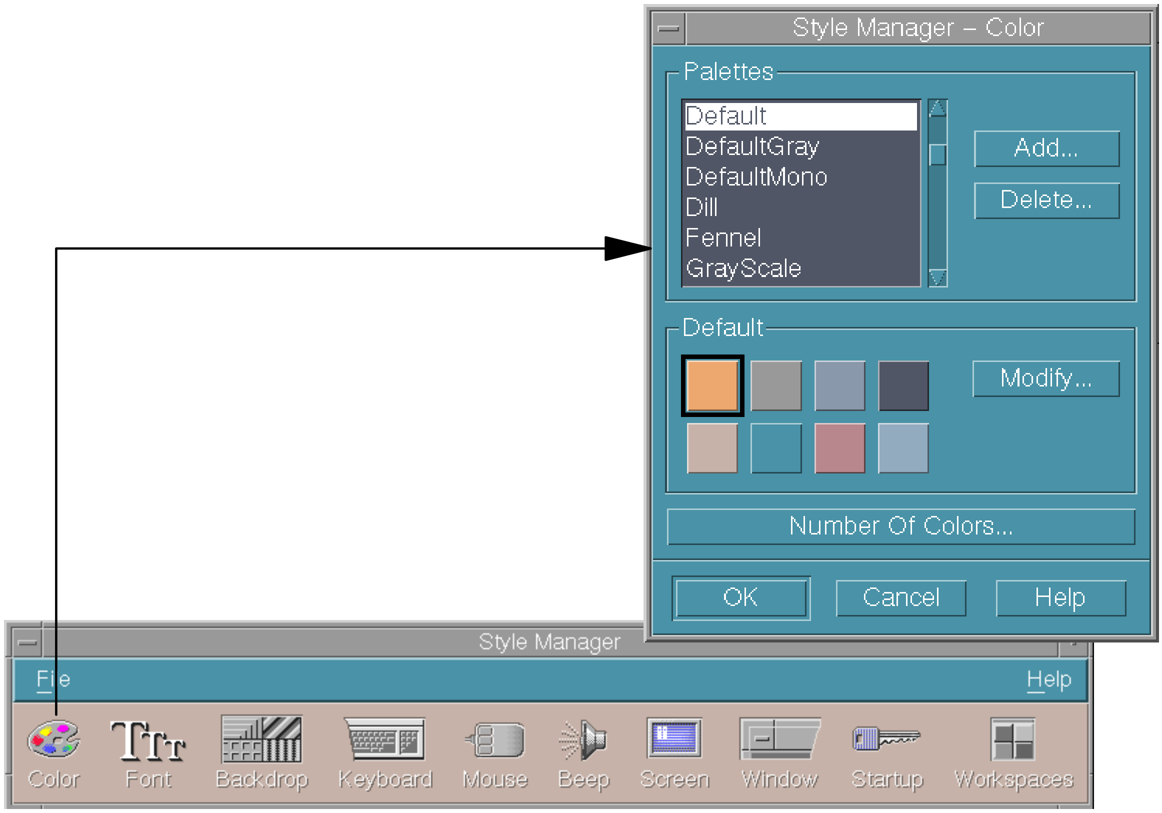


Figure 19-4. Style Manager - Colors AU139.0

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 dialogue 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.

Style Manager - Fonts

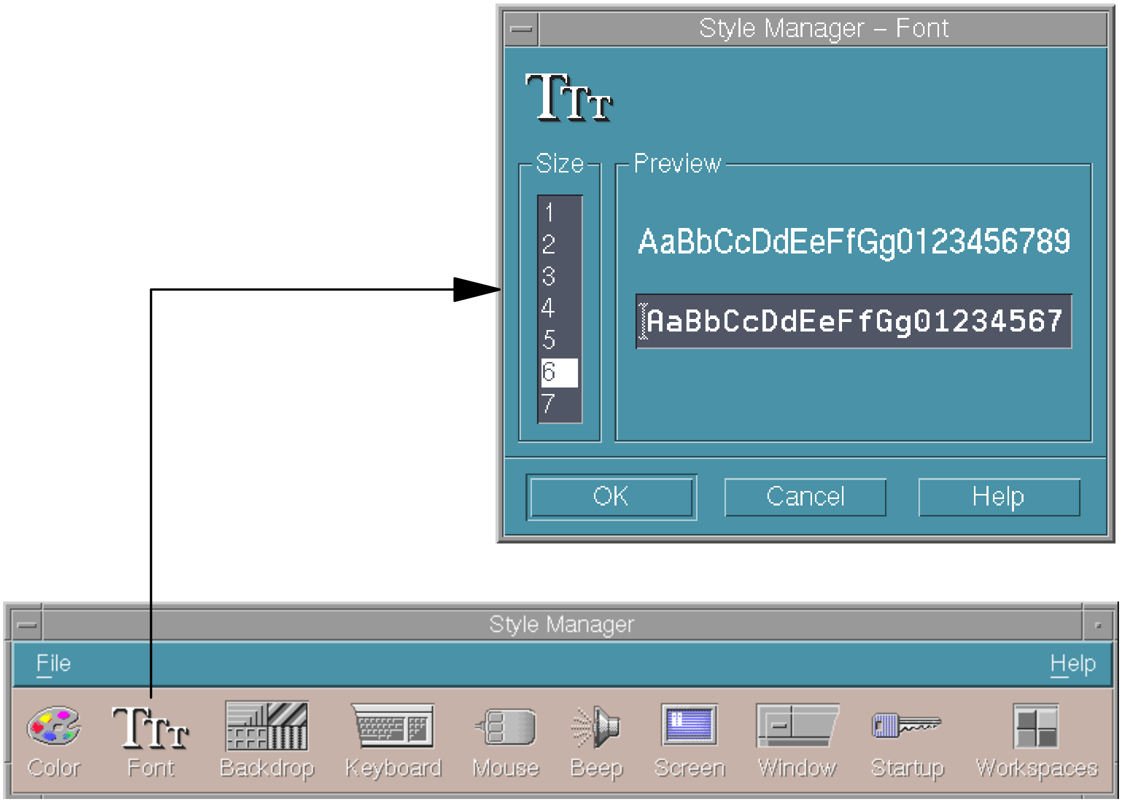


Figure 19-5. Style Manager - Fonts AU139.0

Notes:

Choosing new fonts

The Style Manager font dialogue 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.

Style Manager - Backdrops

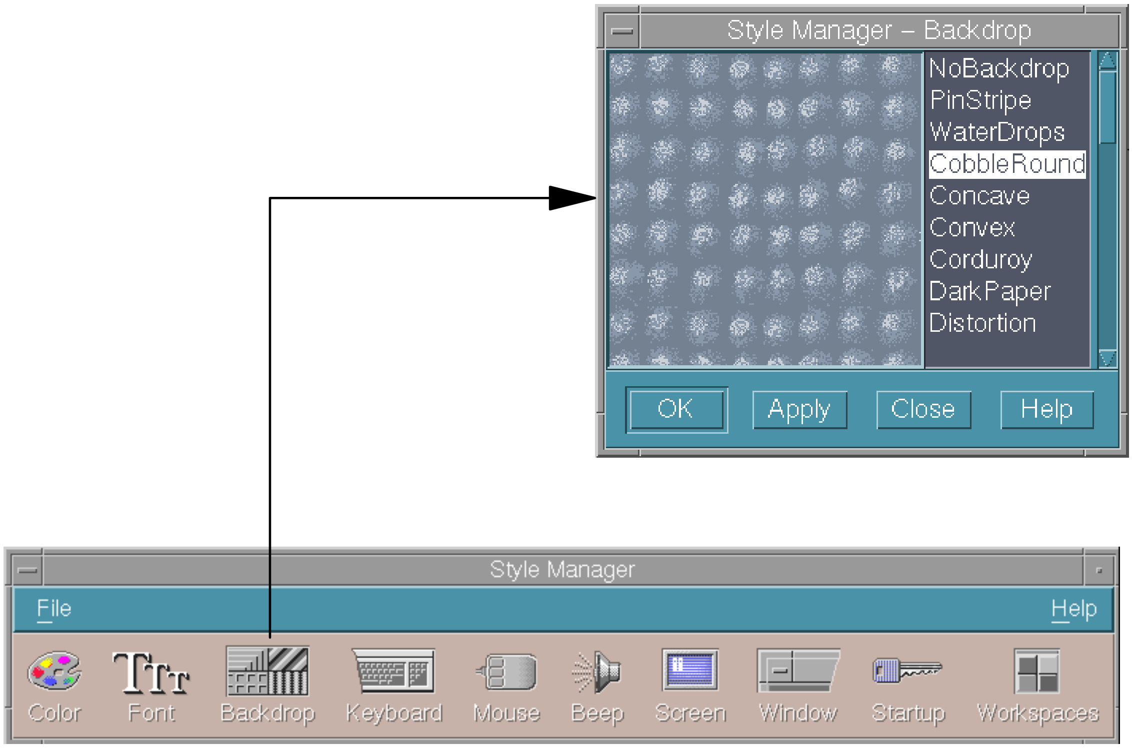


Figure 19-6. Style Manager - Backdrops AU139.0

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.

Style Manager - Keyboard, Mouse and Beep

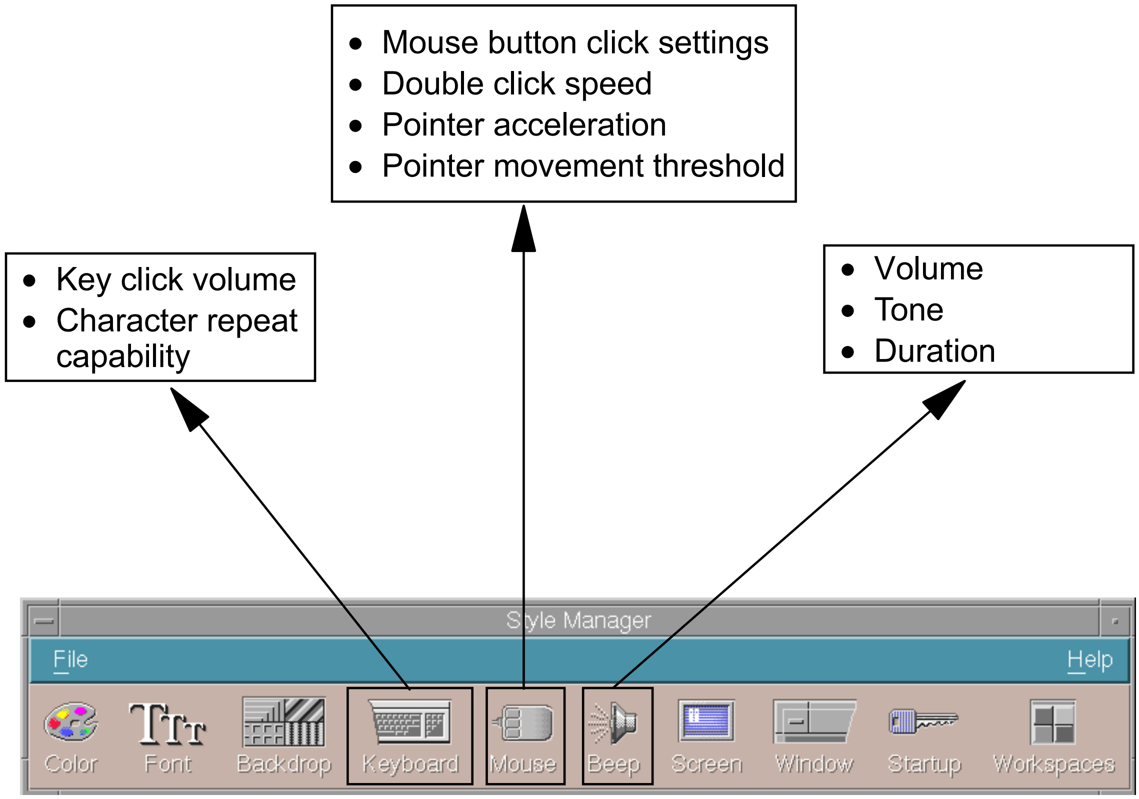


Figure 19-7. Style Manager - Keyboard, Mouse and Beep AU139.0

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.

Style Manager - Window, Screen and Startup

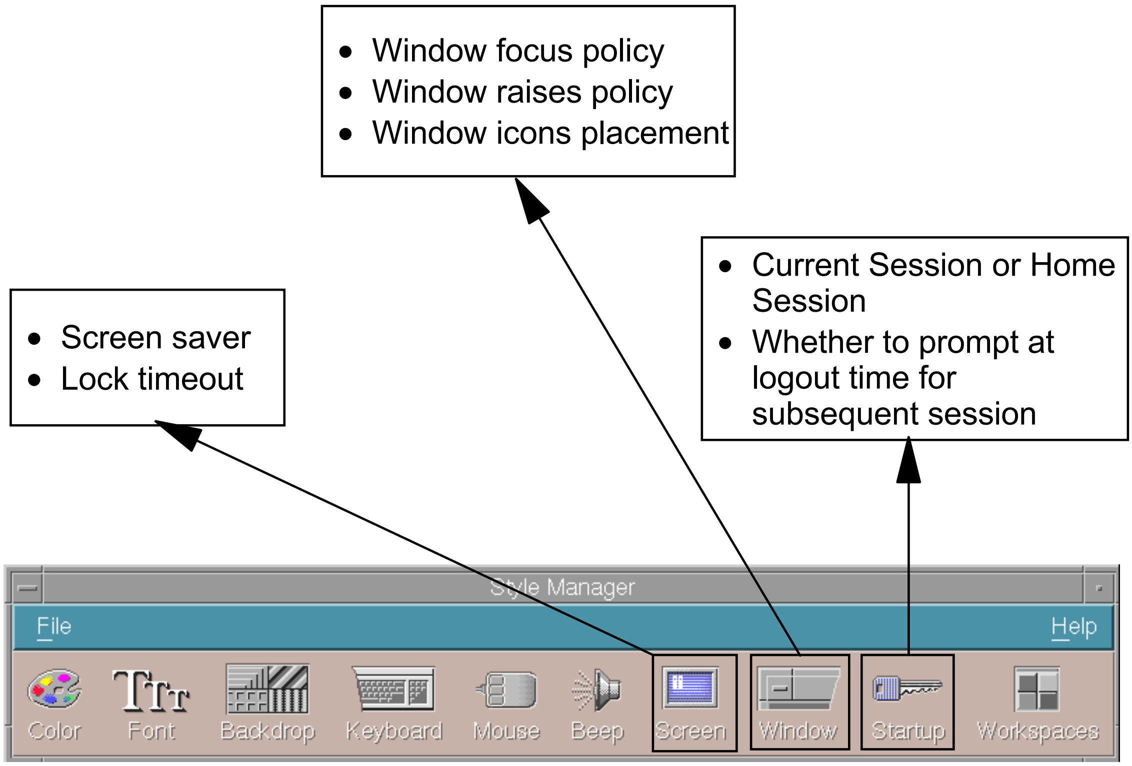


Figure 19-8. Style Manager - Window, Screen and Startup AU139.0

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 dialogue 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 into 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.

Activity: Review Style Manager

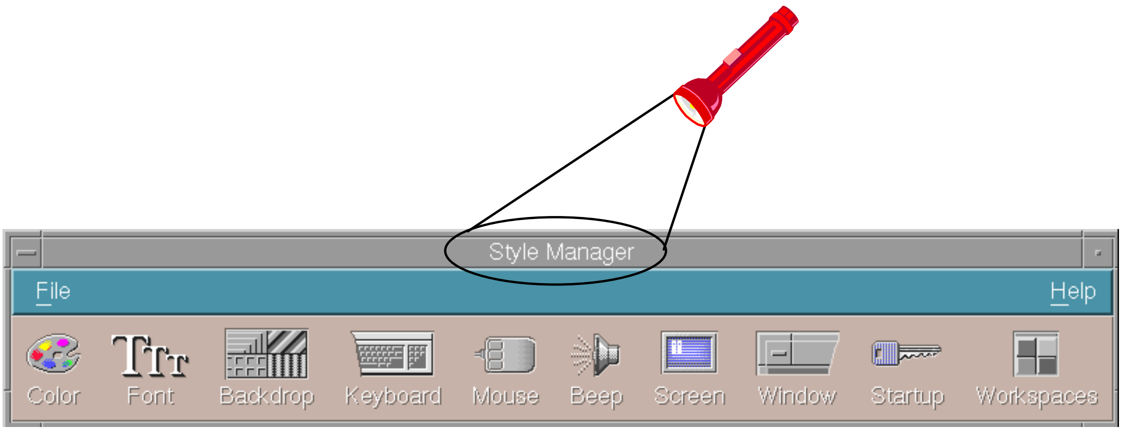


Figure 19-9. Activity: Review Style Manager AU139.0

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.*

General Structure of a Front Panel

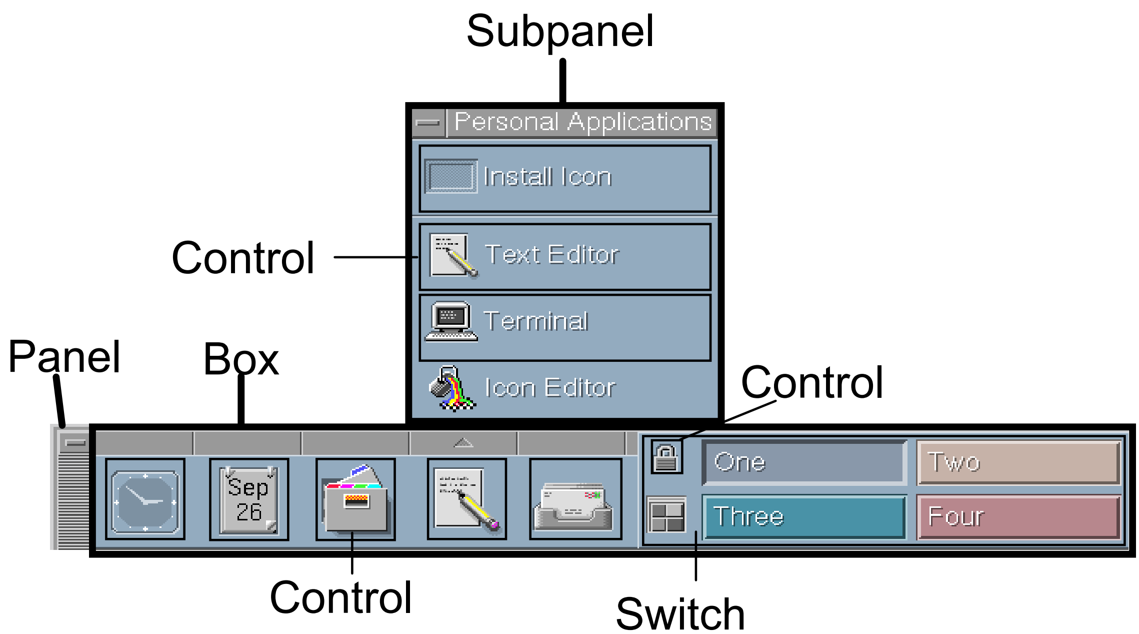


Figure 19-10. General Structure of a Front Panel AU139.0

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.

Creating a New Workspace

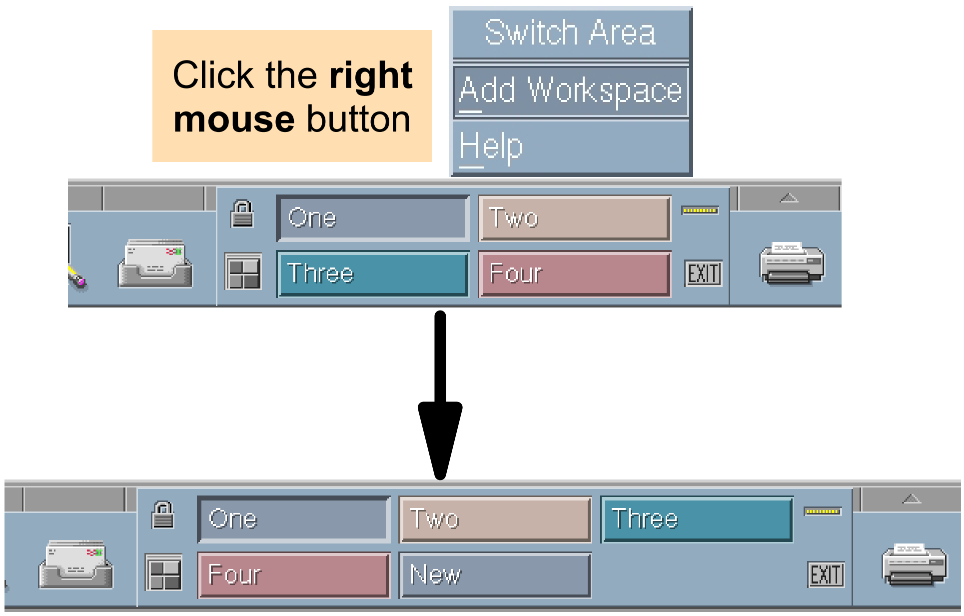


Figure 19-11. Creating a New Workspace AU139.0

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'll 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`

Changing a Workspace Name

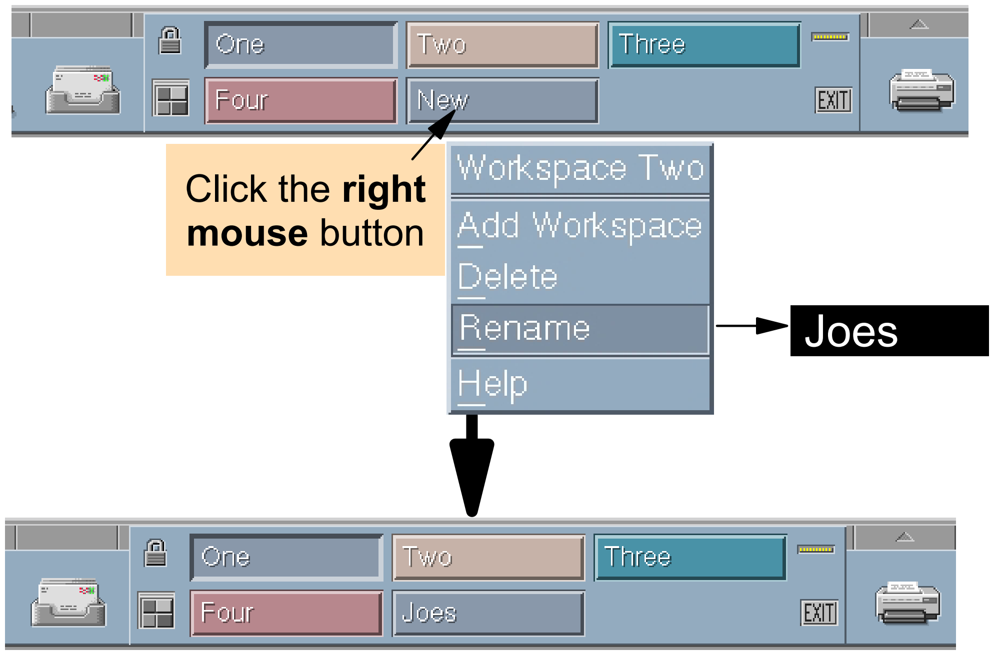


Figure 19-12. Changing a Workspace Name AU139.0

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`.

Dynamic Creation or Deletion of a Subpanel

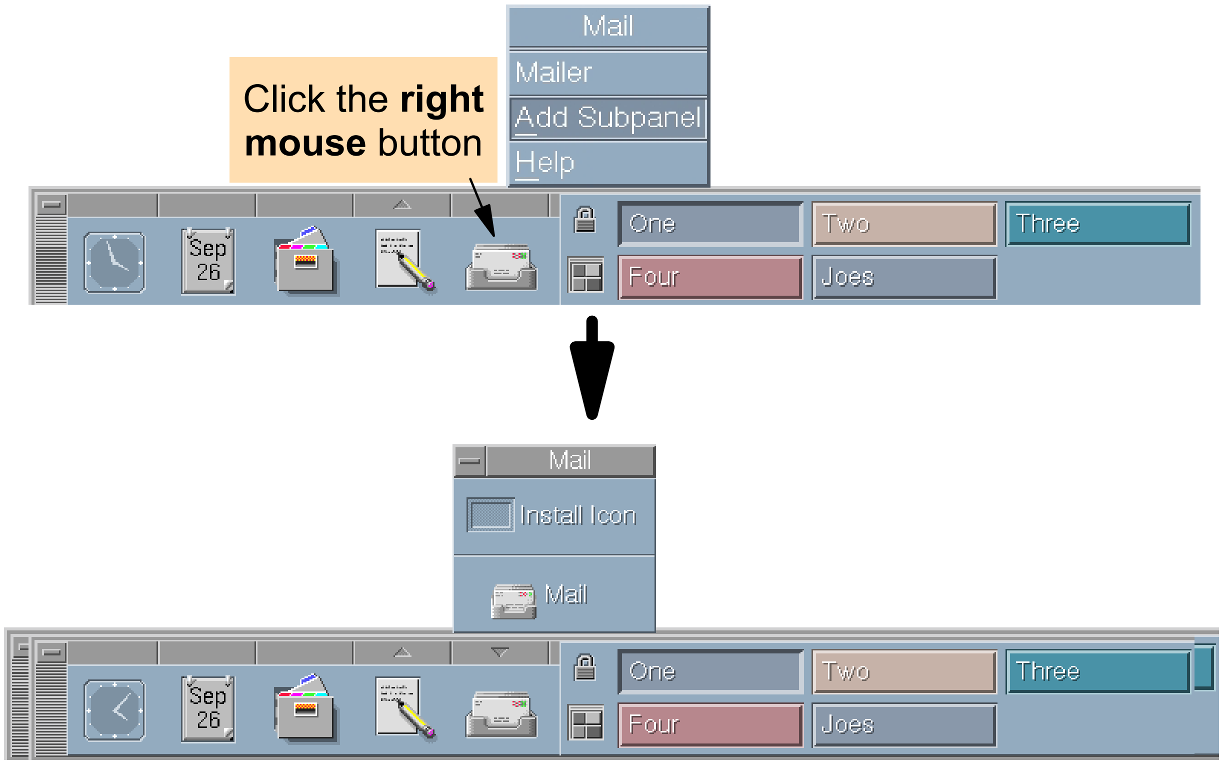


Figure 19-13. Dynamic Creation or Deletion of a Subpanel AU139.0

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.

Adding a Control to a Subpanel

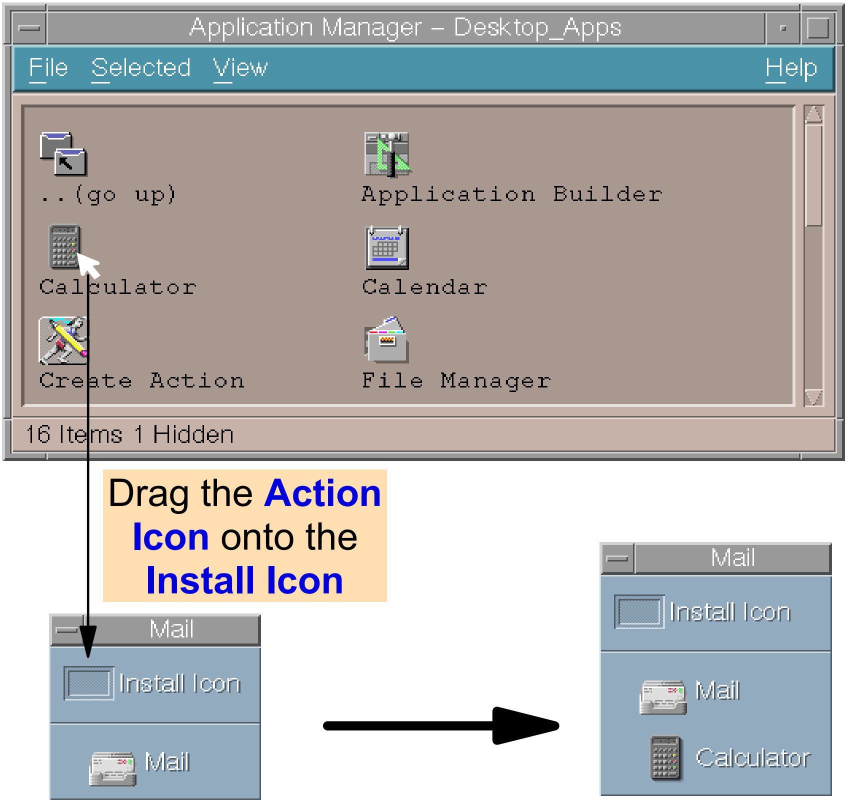


Figure 19-14. Adding a Control to a Subpanel AU139.0

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.

Copy a Subpanel Control to the Main Panel

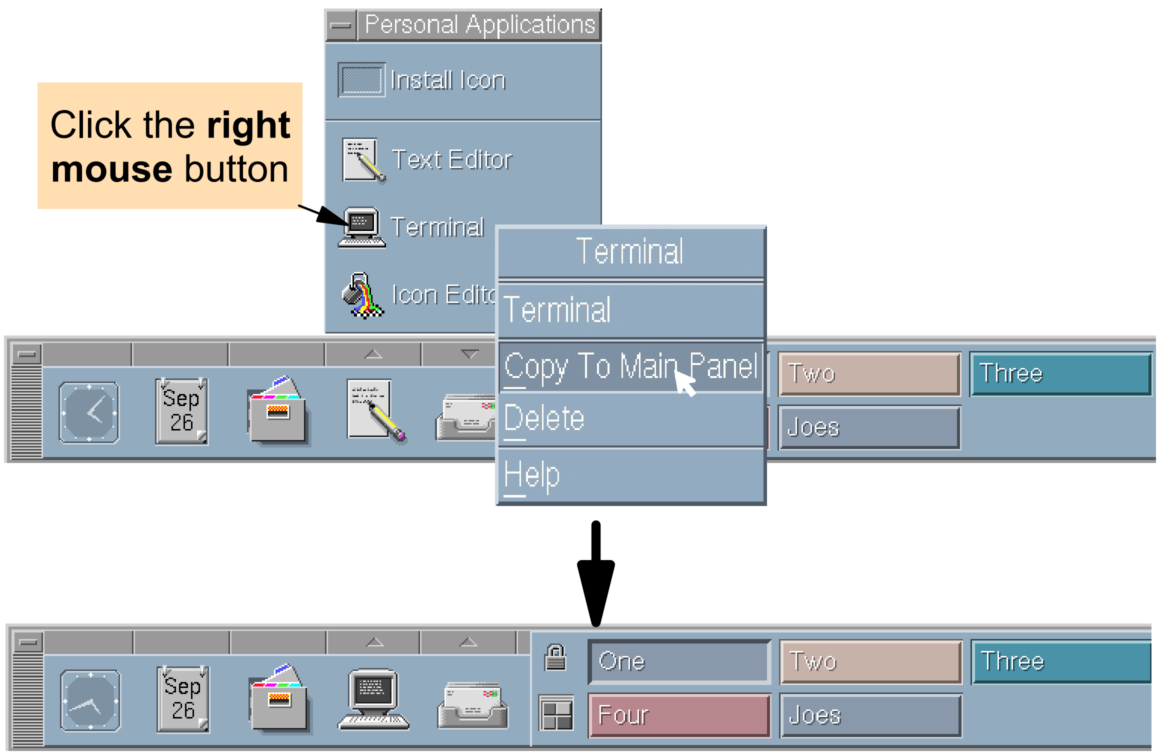


Figure 19-15. Copy a Subpanel Control to the Main Panel AU139.0

Notes:

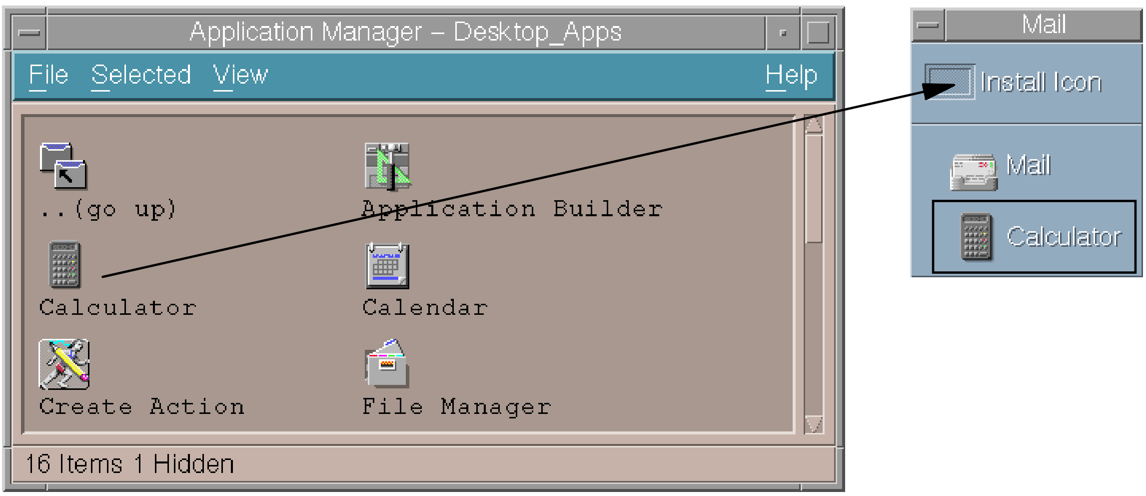
Modifying the Main Panel

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`.

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
```


Figure 19-16. Adding Controls to the Front Panel (1 of 2) AU139.0

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**.

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

Figure 19-17. Adding Controls to the Front Panel (2 of 2) AU139.0

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.

Extended Front Panel

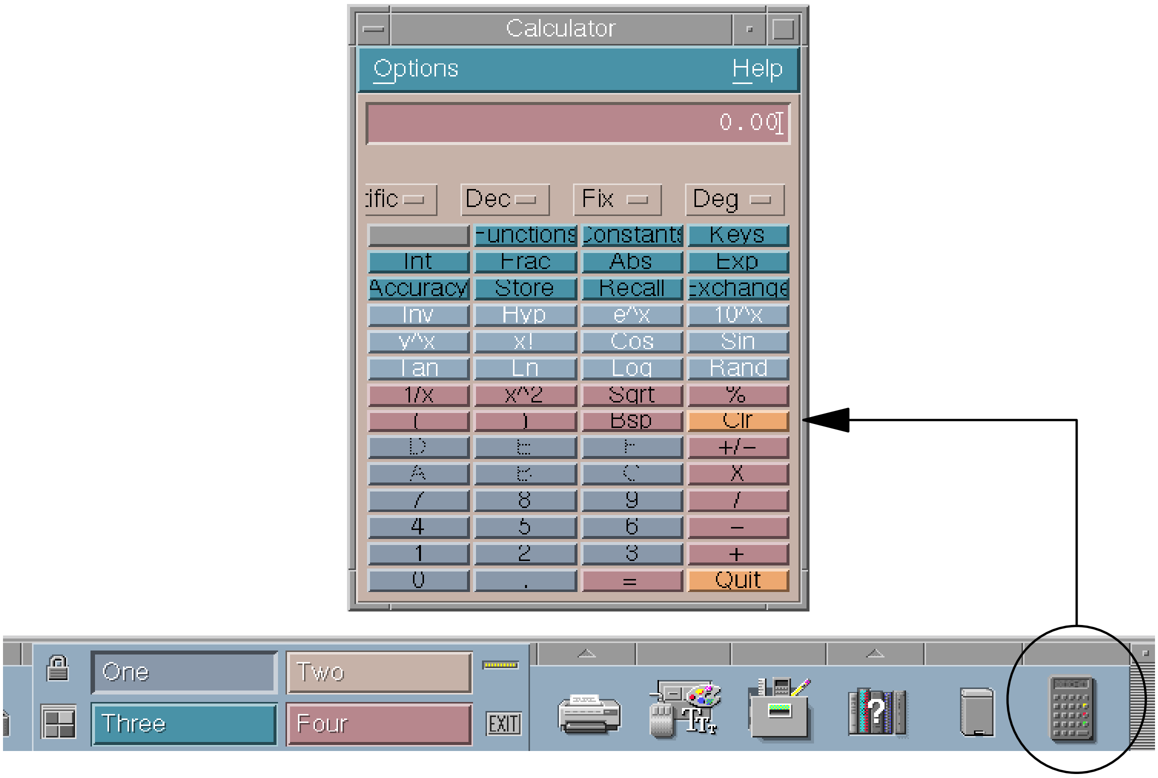


Figure 19-18. Extended Front Panel AU139.0

Notes:

The visual shows how the customized front panel looks after adding the *calculator control.*

Checkpoint

1. How do you customize the screen saver in your desktop environment ?

2. True of 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.

Figure 19-19. Unit 19: Checkpoint AU139.0

Notes:

Exercise: Customizing CDE

Figure 19-20. Exercise: Customizing CDE AU139.0

Notes:

After completing this lab, you will be able to:

- Customize the CDE using the *Style Manager*.
- Customize the *Front Panel*.

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.

Figure 19-21. Unit Summary AU139.0

Notes:

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.

Unit 2: Using the System

Checkpoint Solutions

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? `mail username`

3. What are other commands that can be used to communicate with other users?

   `talk, write, and wall`

4. What output would you expect from the following command: **`cal 8`**?

   The calendar for the year 8 AD

5. Which command would you use to find out when a particular user logged in?

```
$ who am i
$ who
$ finger everyone
$ finger username
```

Unit 3: AIX Documentation

Checkpoint Solutions

1. Which command displays manual entries online?
   `man`
2. Complete the following sentences:
   The AIX 5L 5.3 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`

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**
   Fill 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?
   Move you up two directories
4. What conditions have to be satisfied in order for the `rmdir` command to complete successfully?
   The directory must be empty.
   You must be at least one directory I level higher that the one you are trying to remove.

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	-l	Provides a long listing of files
-i	-a	Will list hidden files
-d	-R	List subdirectories and their contents recursively
-l	-i	Displays the inode number
-R	-d	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

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 know 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`**

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`
4. 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
```

5. 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.

Unit 6: File Permissions (2 of 3)

Checkpoint Solutions (2 of 3)

6. 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
```

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**.

Unit 6: File Permissions (3 of 3)

Checkpoint Solutions (3 of 3)

7. 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
```

Can Fred, who is a member of the finance group, copy the file **joblog** to his home directory? Yes.

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?

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 beginning with any three characters and the fourth character must not be from the range a to z. Then a number of characters can follow, after which the next to last character must be from the range 0 to 9, and the file name must end with a 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):
   keyboard
   screen
   screen

3. `$ mail   tim   <   letter`

   standard input (0):
   standard output (1):
   standard error (2):
   letter
   screen (but mail will not write to stdout in this situation)
   screen

Unit 8: Shell Basics (2 of 2)

Checkpoint Solutions (2 of 2)

4. `$ cat .profile > newprofile 2>1`
   standard input (0):
   standard output (1):
   standard error (2):
   keyboard
   newprofile
   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`

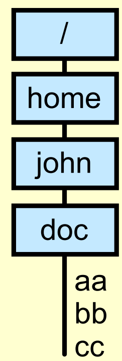
7. Place the output in **fileb** and discard any error messages. (Do not display or store error messages.)
   `$ cat filea > fileb 2> /dev/null`

Unit 9: Using Shell Variables (1 of 2)

Checkpoint Solutions (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**

Unit 9: Using Shell Variables (2 of 2)

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 *
   *

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.

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

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

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).

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
   dircmp
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

Unit 15: Additional Shell Features

Checkpoint Solutions

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
```

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 - i.e. the `TERMTYPE` variable is empty, then again a message will be sent to the user.

2. Write a script that will multiply any two numbers together.

```
expr $1 \* $2
```

Unit 16: AIXwindows Concepts

Checkpoint Solutions

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.

Unit 17: Customizing AIXwindows

Checkpoint Solutions

1. Match the AIXwindows startup file with its function:

a. .xinitrc	b Sets default characteristics for AIXwindows resources
b. .Xdefaults	a Starts the Motif Window Manager
c. .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.**

Unit 18: Using the Common Desktop Environment (CDE)

Checkpoint Solutions

1. True or false? CDE is designed as a common user interface for the UNIX environment. True.

2. Match the following terms with their correct meanings:

a. Login Manager	c	Can be used to obtain a dtterm
b. Front Panel	h	Can be used to work with online documentation
c. Personal Applications	f	Used to customize CDE
d. File Manager	b	The application "Launcher"
e. Application Manager	g	Maintains desktop look between sessions
f. Style Manager	d	Provides a GUI to work with files
g. Session Manager	e	Used to manage applications
h. Help Manager	a	Authenticates the user ID

3. 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.

Unit 19: CDE User Customization

Checkpoint Solutions

1. How do you customize the screen saver in your desktop environment ?
   By using the Style Manager
2. True of 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.

Appendix B. Command Summary

Startup, Logoff and Shutdown

`<Ctrl+d>` **(exit)**	logoff 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. • `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's 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 ul` (files owned by user ul) `-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 don't list) `-l` (list files with matches, but don't list) `-n` (list line numbers with lines) `-v` (find files without pattern) expression metacharacters • [ ] 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, **now='date'** sets the value of `now` to current results of the **date** command.
$	preceding variable name indicates the value of the variable.

Variables

=	set a variable (for example, d=**day** sets the value of `d` to `day`). Can also set the variable to the results of a command by the ' character; for example, **now=date** sets the value of `now` to the current result of the **date** 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, **echo HI** or **echo $d**). 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

format	AIX command to format a diskette
backup	backs up individual files. **-i** reads file names form standard input **-v** list files as backed up; for example, **backup -iv -f/dev/rmto file1, file2** **-u** backup file system at specified level; for example, **backup -level -u filesystem**

	Can pipe list of files to be backed up into command; for example, `find . -print	backup -ivf/dev/rmt0` where you are in directory to be backed up.
`restore`	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 `backup -level -u;` for example, `restore -xv -f/dev/rmt0`

`cpio`	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, `cpio -o > /dev/fd0`
	
	`file1`
	`file2`
	`<Ctrl+d>`
	
	or `cpio -iv file1 < /dev/fd0`

`tar`	alternative utility to backup and restore files
`pax`	alternative utility to cpio and tar commands

Transmitting

`mail`	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: `d` - delete `s` - append `q` - quit `Enter` - skip `m` - forward
`mailx`	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 nl 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: = != (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)
`dW`	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 thru 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

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 dialogue.

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, etc.

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 Pareplace 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 programs 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 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 users 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 cancelling 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®