AIX Version 7.3

Operating system management

Note

Before using this information and the product it supports, read the information in “Notices” on page 397.

This edition applies to AIX Version 7.3 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this document

This document provides users and system administrators with complete information that can affect your selection of options when performing such tasks as backing up and restoring the system, managing physical and logical storage, sizing appropriate paging space, and so on. It provides complete information about how to perform such tasks as managing logical volumes, storage, and resources. System users can learn how to perform such tasks as running commands, handling processes, handling files and directories, and basic printing.

Other topics useful to users and system administrators include creating and resizing paging space, managing virtual memory, backing up and restoring the system, managing hardware and pseudodevices, using the System Resource Controller (SRC), securing files, using storage media, customizing environment files, and writing shell scripts. This document is also available on the documentation CD that is shipped with the operating system.

Highlighting

The following highlighting conventions are used in this document:

Bold	Identifies commands, subroutines, keywords, files, structures, directories, and other items whose names are predefined by the system. Also identifies graphical objects such as buttons, labels, and icons that the user selects.
Italics	Identifies parameters whose actual names or values are to be supplied by the user.
`Monospace`	Identifies examples of specific data values, 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, messages from the system, or information you should actually type.

Case-sensitivity in AIX

Everything in the AIX operating system is case-sensitive, which means that it distinguishes between uppercase and lowercase letters. For example, you can use the **ls** command to list files. If you type LS, the system responds that the command `is not found`. Likewise, **FILEA**, **FiLea**, and **filea** are three distinct file names, even if they reside in the same directory. To avoid causing undesirable actions to be performed, always ensure that you use the correct case.

ISO 9000

ISO 9000 registered quality systems were used in the development and manufacturing of this product.

Operating system management

System administrators and users can learn how to perform such tasks as running commands, handling processes, handling files and directories, backing up and restoring the system, managing physical and logical storage, and basic printing.

Other topics useful to users and system administrators include creating and re-sizing paging space, managing virtual memory, backing up and restoring the system, managing hardware and pseudo devices, using the System Resource Controller (SRC), securing files, using storage media, customizing environment files, and writing shell scripts. This topic is also available on the documentation CD that is shipped with the operating system.

Operating system management is the task of an individual who is usually referred to, in UNIX literature, as the system administrator. Unfortunately, only a few system administrator activities are straightforward enough to be correctly called administration. This and related guides are intended to help system administrators with their numerous duties.

This operating system provides its own particular version of system-management support in order to promote ease of use and to improve security and integrity.

Operating system administration

You can use commands to manage system startup and backup, shutting down the system, system shells and environments, system resources, and other different parts of AIX.

Operating system management is the task of an individual who is usually referred to, in UNIX literature, as the system administrator. Unfortunately, only a few system administrator activities are straightforward enough to be correctly called administration. This and related guides are intended to help system administrators with their numerous duties.

This operating system provides its own particular version of system-management support in order to promote ease of use and to improve security and integrity.

Available system management interfaces

In addition to conventional command line system administration, this operating system provides the SMIT interfaces.

The following are the SMIT interfaces:

- System Management Interface Tool (SMIT), a menu-based user interface that constructs commands from the options you choose and executes them.

  With SMIT, you can:

  - Install, update, and maintain software
  - Configure devices
  - Configure disk storage units into volume groups and logical volumes
  - Make and extend file systems and paging space
  - Manage users and groups
  - Configure networks and communication applications
  - Print
  - Perform problem determination
  - Schedule jobs
  - Manage system resources and workload
  - Manage system environments

  - Manage cluster system data
- An object-oriented graphical user interface that supports the same system management tasks as SMIT, but eases system management tasks by:
  - Reducing user errors through error checking and dialog design
  - Offering step-by-step procedures for new or complex tasks
  - Offering advanced options for more experienced administrators
  - Making it easier to visualize complex data or relationships among system objects
  - Monitoring system activity and alerting the administrator when predefined events occur
  - Providing context-sensitive helps, overviews, tips, and links to online documentation

Software vital product data

Certain information about software products and their installable options is maintained in the Software Vital Product Data (SWVPD) database.

The SWVPD consists of a set of commands and Object Data Manager (ODM) object classes for the maintenance of software product information. The SWVPD commands are provided for the user to query (**lslpp**) and verify (**lppchk**) installed software products. The ODM object classes define the scope and format of the software product information that is maintained.

The **installp** command uses the ODM to maintain the following information in the SWVPD database:

- Name of the installed software product
- Version of the software product
- Release level of the software product, which indicates changes to the external programming interface of the software product
- Modification level of the software product, which indicates changes that do not affect the external programming interface of the software product
- Fix level of the software product, which indicates small updates that are to be built into a regular modification level at a later time
- Fix identification field
- Names, checksums, and sizes of the files that make up the software product or option
- Installation state of the software product: applying, applied, committing, committed, rejecting, or broken.

Operating system updates

The operating system package is divided into filesets, where each fileset contains a group of logically related customer deliverable files. Each fileset can be individually installed and updated.

Revisions to filesets are tracked using the version, release, maintenance, and fix (VRMF) levels. By convention, each time an AIX fileset update is applied, the fix level is adjusted. Each time an AIX maintenance package or technology level is applied, the modification level is adjusted, and the fix level is reset to zero. The initial installation of an AIX version, for example, AIX® 6.1, is called a base installation. The operating system provides updates to its features and functionality, which might be packaged as a maintenance package, a technology level, a program temporary fix (PTF), or a service pack (a group of PTFs).

Maintenance Packages and Technology Levels

Maintenance packages and technology levels provide new functionality that is intended to upgrade the release. The maintenance part of the VRMF is updated in a maintenance package. For example, the first maintenance package for AIX 6.1 is 6.1.1.0; the second is 6.1.2.0, and so on. To list the maintenance package, use the **oslevel -r** command.

To determine the maintenance package or technology level installed on a particular system, type:

```
oslevel
```

To determine which filesets need an update for the system to reach a specific maintenance package or technology level (in this example, 6.1.1.0), use the following command:

```
oslevel -l 6.1.1.0
```

To determine if a recommended maintenance package or technology level is installed (in this example, 6100-02), use the following command:

```
oslevel -r 6100-02
```

To determine which filesets need an update for the system to reach the 6100-02 maintenance package or technology level, use the following command:

```
oslevel -rl 6100-02
```

To determine the maintenance package or technology level of a particular fileset (in this example, `bos.mp`), use the following command:

```
lslpp -L bos.mp
```

PTFs
Between releases, you might receive PTFs to correct or prevent a particular problem. A particular installation might need some, all, or even none of the available PTFs.

Recommended Maintenance Packages
A recommended maintenance package is a set of PTFs between technology levels that have been extensively tested together and are recommended for preventive maintenance.

Interim Fixes
An interim fix is similar to a PTF, but it is usually offered when a PTF is not available. Interim fixes are also released when the PTF would upgrade a system to the next maintenance level and users might want their systems to remain at the current level.

To determine the version and release level, maintenance package, technology level, and service pack level, as well as which filesets need to be updated to reach a particular level, see the oslevel and the lslpp commands .

System startup

When the base operating system starts, the system initiates a complex set of tasks. Under normal conditions, these tasks are performed automatically.

There are some situations when you want to instruct the system to reboot; for example, to cause the system to recognize newly installed software, to reset peripheral devices, to perform routine maintenance tasks like checking file systems, or to recover from a system hang or crash. For information on these procedures, see:

Related tasks
Recreating a corrupted boot image

The following procedure describes how to identify a corrupted boot image and re-create it.

Administering system startup

There are multiple scenarios that you might encounter when you want to boot or reboot your system. To shut down or reboot your system you can use either the shutdown or reboot command. You should use the shutdown command when multiple users are logged on to the system.

Rebooting a running system

Because processes might be running that should be terminated more gracefully than a **reboot** permits, **shutdown** is the preferred method for all systems.

There are two methods for shutting down and rebooting your system, **shutdown** and **reboot**. Always use the **shutdown** method when multiple users are logged on to the system.

Task	SMIT Fast Path	Command or File
Rebooting a Multiuser System	`smit shutdown`	**shutdown -r**
Rebooting a Single-User System	`smit shutdown`	**shutdown -r** or **reboot**

Rebooting a unresponsive system remotely

The remote reboot facility allows the system to be rebooted through a native (integrated) system port.

The POWER5 integrated *system ports* are similar to serial ports except that system ports are available only for specifically supported functions.

The system is rebooted when the **reboot_string** is received at the port. This facility is useful when the system does not otherwise respond but is capable of servicing system port interrupts. Remote reboot can be enabled on only one native system port at a time. Users are expected to provide their own external security for the port. This facility runs at the highest device interrupt class and a failure of the UART (Universal Asynchronous Receive/Transmit) to clear the transmit buffer quickly may have the effect of causing other devices to lose data if their buffers overflow during this time. It is suggested that this facility only be used to reboot a machine that is otherwise hung and cannot be remotely logged into. File systems will *not* be synchronized, and a potential for some loss of data which has not been flushed exists. It is strongly suggested that when remote reboot is enabled that the port not be used for any other purpose, especially file transfer, to prevent an inadvertent reboot.

Two native system port attributes control the operation of remote reboot.

reboot_enable

Indicates whether this port is enabled to reboot the machine on receipt of the remote **reboot_string**, and if so, whether to take a system dump prior to rebooting.

```
no        - Indicates remote reboot is disabled
reboot    - Indicates remote reboot is enabled
dump      - Indicates remote reboot is enabled, and prior to rebooting a system dump
            will be taken on the primary dump device
```

reboot_string

Specifies the remote **reboot_string** that the serial port will scan for when the remote reboot feature is enabled. When the remote reboot feature is enabled and the **reboot_string** is received on the port, a > character is transmitted and the system is ready to reboot. If a 1 character is received, the system is rebooted; any character other than 1 aborts the reboot process. The **reboot_string** has a maximum length of 16 characters and must not contain a space, colon, equal sign, null, new line, or Ctrl-\ character.

Remote reboot can be enabled through SMIT or the command line. For SMIT the path **System Environments -> Manage Remote Reboot Facility** may be used for a configured TTY. Alternatively, when configuring a new TTY, remote reboot may be enabled from the **Add a TTY** or **Change/Show Characteristics of a TTY** menus. These menus are accessed through the path **Devices -> TTY**.

From the command line, the **mkdev** or **chdev** commands are used to enable remote reboot. For example, the following command enables remote reboot (with the dump option) and sets the reboot string to **ReBoOtMe** on **tty1**.

```
chdev -l tty1 -a remreboot=dump -a reboot_string=ReBoOtMe
```

This example enables remote reboot on **tty0** with the current **reboot_string** in the database only (will take effect on the next reboot).

```
chdev -P -l tty0 -a remreboot=reboot
```

If the tty is being used as a normal port, then you will have to use the **pdisable** command before enabling remote reboot. You may use **penable** to reenable the port afterwards.

Related information

Function differences between system ports and serial ports

Booting from hard disk for maintenance

You can boot a machine in maintenance mode from a hard disk.

Prerequisites

A bootable removable media (tape or CD-ROM) must not be in the drive. Also, refer to the hardware documentation for the specific instructions to enable maintenance mode boot on your particular model.

Procedure

To boot a machine in maintenance mode from a hard disk:

1. To reboot, either turn the machine off and then power it back on, or press the reset button.
2. Press the key sequence for rebooting in maintenance mode that is specified in your hardware documentation.
3. The machine will boot to a point where it has a console device configured.

   If there is a system dump that needs to be retrieved, the system dump menu will be displayed on the console.

   Note:

   a. If the console fails to configure when there is a dump to be retrieved, the system will hang. The system must be booted from a removable medium to retrieve the dump.

   b. The system automatically dumps to the specified dump device when the reset button is pressed. To change the primary or secondary dump device designation in a running system, see the **sysdumpdev** command.
4. If there is no system dump, or if it has been copied, the diagnostic operating instructions will be displayed. Press Enter to continue to the **Function Selection** menu.
5. From the **Function Selection** menu, you can select diagnostic or single-user mode:

   `Single-User Mode:` To perform maintenance in a single-user environment, choose this option (option 5). The system continues to boot and enters single-user mode. Maintenance that requires the system to be in a standalone mode can be performed in this mode, and the **bosboot** command can be run, if required.

Related information

Starting a System Dump

Booting a system that has crashed

In some instances, you might have to boot a system that has stopped (crashed) without being properly shut down.

The prerequisites for this procedure are:

- Your system crashed and was not properly shut down due to unusual conditions.

- Your system is turned off.

This procedure covers the basics of how to boot if your system was unable to recover from a crash. Perform the following steps:

1. Ensure that all hardware and peripheral devices are correctly connected.
2. Turn on all of the peripheral devices.
3. Watch the screen for information about automatic hardware diagnostics.
   a) If any hardware diagnostics tests are unsuccessful, refer to the hardware documentation.
   b) If all hardware diagnostics tests are successful, turn the system unit on.

Resetting an unknown root password

The following procedure describes how to recover access to root privileges when the system's root password is unavailable or unknown.

The following procedure requires some system downtime. If possible, schedule your downtime when it least impacts your workload to protect yourself from a possible loss of data or functionality.

The information in this how-to scenario was tested using specific versions of AIX. The results you obtain might vary significantly depending on your version and level of AIX.

1. Insert the product media for the same version and level as the current installation into the appropriate drive.
2. Power on the machine.
3. When the screen of icons appears, or when you hear a double beep, press the F1 key repeatedly until the **System Management Services** menu appears.
4. Select **Multiboot**.
5. Select **Install From**.
6. Select the device that holds the product media and then select **Install**.
7. Select the AIX version icon.
8. Define your current system as the system console by pressing the F1 key and then press Enter.
9. Select the number of your preferred language and press Enter.
10. Choose **Start Maintenance Mode for System Recovery** by typing 3 and press Enter.
11. Select **Access a Root Volume Group**.

    A message displays explaining that you will not be able to return to the Installation menus without rebooting if you change the root volume group at this point.
12. Type `0` and press Enter.
13. Type the number of the appropriate volume group from the list and press Enter.
14. Select **Access this Volume Group and start a shell** by typing 1 and press Enter.
15. At the # (number sign) prompt, type the **passwd** command at the command line prompt to reset the root password.

    For example:

    ```
    # passwd
    Changing password for "root"
    root's New password:
    Enter the new password again:
    ```
16. To write everything from the buffer to the hard disk and reboot the system, type the following:

    ```
    sync;sync;sync;reboot
    ```

When the login screen appears, the password you set in step 15 should now permit access to root privileges.

Related information

passwd command

reboot command

Booting systems with planar graphics

If the machine has been installed with the planar graphics susbsystem only, and later an additional graphics adapter is added to the system, the following occurs:

1. A new graphics adapter is added to the system, and its associated device driver software is installed.
2. The system is rebooted, and one of the following occurs:
   a. If the system console is defined to be `/dev/lft0` (**lscons** displays this information), the user is asked to select which display is the system console at reboot time. If the user selects a graphics adapter (non-TTY device), it also becomes the new default display. If the user selects a TTY device instead of an LFT device, no system login appears. Reboot again, and the TTY login screen is displayed. It is assumed that if the user adds an additional graphics adapter into the system and the system console is an LFT device, the user will not select the TTY device as the system console.
   b. If the system console is defined to be a TTY, then at reboot time the newly added display adapter becomes the default display.

      Note: Since the TTY is the system console, it remains the system console.
3. If the system console is `/def/lft0`, then after reboot, DPMS is disabled in order to show the system console selection text on the screen for an indefinite period of time. To re-enable DPMS, reboot the system again.

Deploying run level script execution

Run level scripts allow users to start and stop selected applications while changing the run level.

Put run level scripts in the subdirectory of `/etc/rc.d` that is specific to the run level:

- `/etc/rc.d/rc2.d`
- `/etc/rc.d/rc3.d`
- `/etc/rc.d/rc4.d`
- `/etc/rc.d/rc5.d`
- `/etc/rc.d/rc6.d`
- `/etc/rc.d/rc7.d`
- `/etc/rc.d/rc8.d`
- `/etc/rc.d/rc9.d`

The `/etc/rc.d/rc` will run the scripts it finds in the specified directory when the run level changes - first running the stop application scripts then running the start application scripts.

Note: Scripts beginning with K are stop scripts, while scripts beginning with S are start scripts.

Modifying the /etc/inittab file

Four commands are available to modify the records in the `etc/inittab` file.

Adding records - `mkitab` command

To add a record to the `/etc/inittab` file, type the following at a command prompt:

```
mkitab Identifier:Run Level:Action:Command
```

For example, to add a record for tty2, type the following at a command prompt:

```
mkitab tty002:2:respawn:/usr/sbin/getty /dev/tty2
```

In the above example:

Item	Description
`tty002`	Identifies the object whose run level you are defining.

Item	Description
2	Specifies the run level at which this process runs.
`respawn`	Specifies the action that the **init** command should take for this process.
`/usr/sbin/getty /dev/tty2`	Specifies the shell command to be executed.

Changing records - `chitab` command

To change a record to the **/etc/inittab** file, type the following at a command prompt:

```
chitab Identifier:Run Level:Action:Command
```

For example, to change a record for tty2 so that this process runs at run levels 2 and 3, type:

```
chitab tty002:23:respawn:/usr/sbin/getty /dev/tty2
```

In the above example:

Item	Description
`tty002`	Identifies the object whose run level you are defining.
23	Specifies the run levels at which this process runs.
`respawn`	Specifies the action that the **init** command should take for this process.
`/usr/sbin/getty /dev/tty2`	Specifies the shell command to be executed.

Listing records - `lsitab` command

To list all records in the `/etc/inittab` file, type the following at a command prompt:

```
lsitab -a
```

To list a specific record in the `/etc/inittab` file, type:

```
lsitab Identifier
```

For example, to list the record for tty2, type: `lsitab tty2`.

Removing records - `rmitab` command

To remove a record from the `/etc/inittab` file, type the following at a command prompt:

```
rmitab Identifier
```

For example, to remove the record for tty2, type: `rmitab tty2`.

Related concepts

System run level

The system run level specifies the system state and defines which processes are started.

Reactivation of an inactive system

Your system can become inactive because of a hardware problem, a software problem, or a combination of both.

This procedure guides you through steps to correct the problem and restart your system. If your system is still inactive after completing the procedure, refer to the problem-determination information in your hardware documentation.

Use the following procedures to reactivate an inactive system:

Hardware check
There are several procedures you can use to check your hardware.

Check your hardware by:

Checking the power:

If the Power-On light on your system is active, go to **Checking the operator panel display**, below.

If the Power-On light on your system is not active, check that the power is on and the system is plugged in.

Checking the operator panel display:

If your system has an operator panel display, check it for any messages.

If the operator panel display on your system is blank, go to **Activating your display or terminal**, below.

If the operator panel display on your system is not blank, go to the service guide for your unit to find information concerning digits in the Operator Panel Display.

Activating your display or terminal:

Check several parts of your display or terminal, as follows:

- Make sure the display cable is securely attached to the display and to the system unit.
- Make sure the keyboard cable is securely attached.
- Make sure the mouse cable is securely attached.
- Make sure the display is turned on and that its Power-On light is lit.
- Adjust the brightness control on the display.
- Make sure the terminal's communication settings are correct.

If your system is now active, your hardware checks have corrected the problem.

Related tasks
Restarting the system
In addition to checking the hardware and checking the processes, you can restart you system to reactivate an inactive system.

Checking the processes
A stopped or stalled process might make your system inactive.

Checking the processes
A stopped or stalled process might make your system inactive.

Check your system processes by:

1. Restarting line scrolling
2. Using the Ctrl+D key sequence
3. Using the Ctrl+C key sequence
4. Logging in from a remote terminal or host
5. Ending stalled processes remotely

Restarting line scrolling:

Restart line scrolling halted by the Ctrl-S key sequence by doing the following:

1. Activate the window or shell with the problem process.
2. Press the Ctrl-Q key sequence to restart scrolling.
   The Ctrl-S key sequence stops line scrolling, and the Ctrl-Q key sequence restarts line scrolling.

If your scroll check did not correct the problem with your inactive system, go to the next section, **Using the Ctrl-D key sequence**.

Using the Ctrl-D key sequence:

1. Activate the window or shell with the problem process.
2. Press the Ctrl-D key sequence. The Ctrl-D key sequence sends an end of file (EOF) signal to the process. The Ctrl-D key sequence may close the window or shell and log you out.

If the Ctrl-D key sequence did not correct the problem with your inactive system, go to the next section, **Using the Ctrl-C key sequence**.

Using the Ctrl-C key sequence:

End a stopped process by doing the following:

1. Activate the window or shell with the problem process.
2. Press the Ctrl-C key sequence. The Ctrl-C key sequence stops the current search or filter.

If the Ctrl-C key sequence did not correct the problem with your inactive system, go to the next section, **Logging in from a remote terminal or host:**.

Logging in from a remote terminal or host:

Log in remotely in either of two ways:

- Log in to the system from another terminal if more than one terminal is attached to your system.
- Log in from another host on the network (if your system is connected to a network) by typing the **tn** command as follows:

```
tn YourSystemName
```

  The system asks for your regular login name and password when you use the **tn** command.

If you were able to log in to the system from a remote terminal or host, go to the next section, **Ending stalled processes remotely**.

If you were not able to log in to the system from a remote terminal or host you need to restart the system.

You can also start a system dump to determine why your system became inactive.

Ending stalled processes remotely:

End a stalled process from a remote terminal by doing the following:

1. List active processes by typing the following **ps** command:

```
ps -ef
```

   The **-e** and **-f** flags identify all active and inactive processes.
2. Identify the process ID of the stalled process.

   For help in identifying processes, use the **grep** command with a search string. For example, to end the **xlock** process, type the following to find the process ID:

```
ps -ef | grep xlock
```

   The **grep** command allows you to search on the output from the **ps** command to identify the process ID of a specific process.
3. End the process by typing the following **kill** command:

   Note: You must have root user authority to use the **kill** command on processes you did not initiate.

```
kill -9 ProcessID
```

   If you cannot identify the problem process, the most recently activated process might be the cause of your inactive system. End the most recent process if you think that is the problem.

If your process checks have not corrected the problem with your inactive system you need to restart the system.

Related concepts

Hardware check

There are several procedures you can use to check your hardware.

Related tasks

Restarting the system

In addition to checking the hardware and checking the processes, you can restart you system to reactivate an inactive system.

Related information

System Dump Facility

Restarting the system

In addition to checking the hardware and checking the processes, you can restart you system to reactivate an inactive system.

If the procedures for “Hardware check” on page 9 and “Checking the processes” on page 9 fail to correct the problem that makes your system inactive, you need to restart your system.

Note: Before restarting your system, complete a system dump.

1. Check the state of the boot device.

   Your system boots with either a removable medium, an external device, a small computer system interface (SCSI) device, an integrated device electronics (IDE) device, or a local area network (LAN). Decide which method applies to your system, and use the following instructions to check the boot device:

   - For a removable medium, such as tape, make sure the medium is inserted correctly.
   - For IDE devices, verify that the IDE device ID settings are unique per adapter. If only one device is attached to the adapter, the IDE device ID must be set to the master device.
   - For an externally attached device, such as a tape drive, make sure:
     - The power to the device is turned on.
     - The device cables are correctly attached to the device and to the system unit.
     - The ready indicator is on (if the device has one).
   - For external SCSI devices, verify that the SCSI address settings are unique.
   - For a LAN, verify that the network is up and operable.

   If the boot device is working correctly, continue to the next step.
2. Load your operating system by doing the following:
   a) Turn off your system's power.
   b) Wait one minute.
   c) Turn on your system's power.
   d) Wait for the system to boot.

If the operating system failed to load, boot the hard disk from maintenance mode or hardware diagnostics.

If you are still unable to restart the system, use an SRN to report the problem with your inactive system to your service representative.

Related concepts

Hardware check

There are several procedures you can use to check your hardware.

Related tasks

Checking the processes

A stopped or stalled process might make your system inactive.

Related information
System Dump Facility

Creating boot images

To install the base operating system or to access a system that will not boot from the system hard drive, you need a boot image. This procedure describes how to create boot images. The boot image varies for each type of device.

When the system is first installed, the **bosboot** command creates a boot image from a RAM (random access memory) disk file system image and the operating system kernel. The boot image is transferred to a particular media such as the hard disk. When the machine is rebooted, the boot image is loaded from the media into memory. For more information about the **bosboot** command, see **bosboot**.

The associated RAM disk file system contains device configuration routines for the following devices:

- `Disk`
- `Tape`
- `CD-ROM`
- `Network Token-Ring, Ethernet, or FDDI device`
- You must have root user authority to use the **bosboot** command.
- The `/tmp` file system must have at least 20 MB of free space.
- The physical disk must contain the boot logical volume. To determine which disk device to specify, type the following at a command prompt:

  ```
  lsvg -l rootvg
  ```

  The **lsvg -l** command lists the logical volumes on the root volume group (rootvg). From this list you can find the name of the boot logical volume.

  Then type the following at a command prompt:

  ```
  lsvg -M rootvg
  ```

  The **lsvg -M** command lists the physical disks that contain the various logical volumes.

Creating a boot image on a boot logical volume

If the base operating system is being installed (either a new installation or an update), the **bosboot** command is called to place the boot image on the boot logical volume. The boot logical volume is a physically contiguous area on the disk created through the Logical Volume Manager (LVM) during installation.

For a list of prerequisites for this procedure, see "Creating boot images" on page 12.

The **bosboot** command does the following:

1. Checks the file system to see if there is enough room to create the boot image.
2. Creates a RAM file system using the **mkfs** command and a prototype file.
3. Calls the **mkboot** command, which merges the kernel and the RAM file system into a boot image.
4. Writes the boot image to the boot logical volume.

To create a boot image on the default boot logical volume on the fixed disk, type the following at a command prompt:

```
bosboot -a
```

OR:

```
bosboot -ad /dev/ipldevice
```

Note: Do not reboot the machine if the **bosboot** command fails while creating a boot image. Resolve the problem and run the **bosboot** command to successful completion.

You must reboot the system for the new boot image to be available for use.

Creating boot images for network devices

You can create boot images for an Ethernet boot or Token-Ring boot.

For a list of prerequisites for this procedure, see "Creating boot images" on page 12.

To create a boot image for an Ethernet boot, type the following at a command prompt:

```
bosboot -ad /dev/ent
```

For a Token-Ring boot:

```
bosboot -ad /dev/tok
```

System run level

The system run level specifies the system state and defines which processes are started.

For example, when the system run level is 3, all processes defined to operate at that run level are started. Near the end of the system boot phase of the boot process, the run level is read from the initdefault entry of the `/etc/inittab` file. The system operates at that run level until it receives a signal to change it. The system run level can be changed with the **init** command. The `/etc/inittab` file contains a record for each process that defines run levels for that process. When the system boots, the **init** command reads the `/etc/inittab` file to determine which processes to start.

The following are the currently-defined run levels:

Item	Description
0-9	When the **init** command changes to run levels 0-9, it kills all processes at the current run levels then restarts any processes associated with the new run levels.
0-1	Reserved for the future use of the operating system.
2	Default run level.
3-9	Can be defined according to the user's preferences.
a, b, c	When the **init** command requests a change to run levels **a**, **b**, or **c**, it does not kill processes at the current run levels; it simply starts any processes assigned with the new run levels.
Q, q	Tells the **init** command to reexamine the `/etc/inittab` file.

Related tasks

Modifying the /etc/inittab file
Four commands are available to modify the records in the `etc/inittab` file.

Identifying the system run level

Before performing maintenance on the operating system or changing the system run level, you might need to examine the various run levels.

This procedure describes how to identify the run level at which the system is operating and how to display a history of previous run levels. The **init** command determines the system run level.

Identification of the current run level

At the command line, type `cat /etc/.init.state`. The system displays one digit; that is the current run level. See the **init** command or the `/etc/inittab` file for more information about run levels.

Displaying a history of previous run levels
You can display a history of previous run levels using the **fwtmp** command.

Note: The `bosext2.acct.obj` code must be installed on your system to use this command.

1. Log in as root user.
2. Type the following at a command prompt:

```
/usr/lib/acct/fwtmp </var/adm/wtmp |grep run-level
```

The system displays information similar to the following:

```
run-level 2  0 1 0062 0123 697081013 Sun Feb  2 19:36:53 CST 1992
run-level 2  0 1 0062 0123 697092441 Sun Feb  2 22:47:21 CST 1992
run-level 4  0 1 0062 0123 698180044 Sat Feb 15 12:54:04 CST 1992
run-level 2  0 1 0062 0123 698959131 Sun Feb 16 10:52:11 CST 1992
run-level 5  0 1 0062 0123 698967773 Mon Feb 24 15:42:53 CST 1992
```

Configuring run levels on multiuser systems
You can change run levels on multiuser systems.

1. Check the `/etc/inittab` file to confirm that the run level to which you are changing supports the processes that you are running.

   The getty process is particularly important, since it controls the terminal line access for the system console and other logins. Ensure that the getty process is enabled at all run levels.
2. Use the **wall** command to inform all users that you intend to change the run level and request that users log off.

   For more information about the **wall** command, see **wall**.
3. Use the **smit telinit** fast path to access the **Set System Run Level** menu.
4. Type the new run level in the **System RUN LEVEL** field.
5. Press Enter to implement all of the settings in this procedure.

   The system responds by telling you which processes are terminating or starting as a result of the change in run level and by displaying the message:

```
INIT: New run level: n
```

   where *n* is the new run-level number.

Configuring run levels on single-user systems
You can change run levels on single-user systems.

1. Check the `/etc/inittab` file to confirm that the run level to which you are changing supports the processes that you are running.

   The getty process is particularly important, since it controls the terminal line access for the system console and other logins. Ensure that the getty process is enabled at all run levels. For more information about the `inittab` file, see `inittab`.
2. Use the `smit telinit` fast path to access the **Set System Run Level** menu.

   For more information about the `telinit` command, see `telinit`.
3. Type the new system run level in the **System RUN LEVEL** field.
4. Press Enter to implement all of the settings in this procedure.

   The system responds by telling you which processes are terminating or starting as a result of the change in run level and by displaying the message:

```
INIT: New run level: n
```

   where *n* is the new run-level number.

Boot process

There are three types of system boots and two resources that are required in order to boot the operating system.

During the boot process, the system tests the hardware, loads and runs the operating system, and configures devices. To boot the operating system, the following resources are required:

- A *boot image* that can be loaded after the machine is turned on or reset.
- Access to the root (/) and `/usr` file systems.

There are three types of system boots:

Item	Description
Hard Disk Boot	A machine is started for normal operations.
Diskless Network Boot	A diskless or dataless workstation is started remotely over a network. A machine is started for normal operations. One or more remote file servers provide the files and programs that diskless or dataless workstations need to boot.
Maintenance Boot	A machine is started from a hard disk, network, tape, or CD-ROM in maintenance mode. A system administrator can perform tasks such as installing new or updated software and running diagnostic checks.

During a hard disk boot, the boot image is found on a local disk created when the operating system was installed. During the boot process, the system configures all devices found in the machine and initializes other basic software required for the system to operate (such as the Logical Volume Manager). At the end of this process, the file systems are mounted and ready for use.

The same general requirements apply to diskless network clients. They also require a boot image and access to the operating system file tree. Diskless network clients have no local file systems and get all their information by way of remote access.

Related concepts

Processing the system boot
Most users perform a hard disk boot when starting the system for general operations. The system finds all information necessary to the boot process on its disk drive.

Maintenance boot process
Occasions might arise when a boot is needed to perform special tasks such as installing new or updated software, performing diagnostic checks, or for maintenance. In this case, the system starts from a bootable medium such as a CD-ROM, DVD, tape drive, network, or disk drive.

RAM file system
The RAM file system, part of the boot image, is totally memory-resident and contains all programs that allow the boot process to continue. The files in the RAM file system are specific to the type of boot.

Processing the system boot

Most users perform a hard disk boot when starting the system for general operations. The system finds all information necessary to the boot process on its disk drive.

When the system is started by turning on the power switch (a cold boot) or restarted with the **reboot** or **shutdown** commands (a warm boot), a number of events must occur before the system is ready for use. These events can be divided into the following phases:

Related concepts

Boot process

There are three types of system boots and two resources that are required in order to boot the operating system.

Firmware phase
The firmware prepares the system to load and run the operating system.

Its initialization phase involves the following steps:

1. The firmware performs basic testing on the system resources that are required for starting the operating system.
2. The firmware checks the user boot list, a list of available boot devices. This boot list can be changed to suit your requirements by using the **bootlist** command. If the user boot list in non-volatile random access memory (NVRAM) is not valid or if a valid boot device is not found, the default boot list is then checked. In either case, the first valid boot device found in the boot list is used for system startup. If a valid user boot list exists in NVRAM, the devices in the list are checked in order. If no user boot list exists, all adapters and devices on the bus are checked. In either case, devices are checked in a continuous loop until a valid boot device is found for system startup.

   Note: The system maintains a default boot list that is stored in NVRAM for normal mode boot. A separate service mode boot list is also stored in NVRAM, and you must refer to the specific hardware instructions for your model to learn how to access the service-mode boot list.
3. When a valid boot device is found, the first record or program sector number (PSN) is checked. If it is a valid boot record, it is read into memory and is added to the IPL control block in memory. Included in the key boot record data are the starting location of the boot image on the boot device, the length of the boot image, and instructions on where to load the boot image in memory.
4. The boot image is read sequentially from the boot device into memory starting at the location specified in NVRAM. The disk boot image consists of the kernel, a RAM file system, and base customized device information.
5. Control is passed to the kernel, which begins system initialization.
6. The kernel runs **init**, which runs phase 1 of the `rc.boot` script.

When the kernel initialization phase is completed, base device configuration begins.

Base device configuration phase
The **init** process starts the `rc.boot` script. Phase 1 of the `rc.boot` script performs the base device configuration.

Phase 1 of the `rc.boot` script includes the following steps:

1. The boot script calls the **restbase** program to build the customized Object Data Manager (ODM) database in the RAM file system from the compressed customized data.
2. The boot script starts the configuration manager, which accesses phase 1 ODM configuration rules to configure the base devices.
3. The configuration manager starts the **sys**, **bus**, **disk**, SCSI, and the Logical Volume Manager (LVM) and rootvg volume group configuration methods.
4. The configuration methods load the device drivers, create special files, and update the customized data in the ODM database.

Booting the system
This procedure completes the system boot phase.

1. The **init** process starts phase 2 running of the `rc.boot` script. Phase 2 of `rc.boot` includes the following steps:
   a) Call the **ipl_varyon** program to vary on the rootvg volume group.
   b) Mount the hard disk file systems onto their normal mount points.
   c) Run the **swapon** program to start paging.

d) Copy the customized data from the ODM database in the RAM file system to the ODM database in the hard disk file system.

e) Exit the `rc.boot` script.

2. After phase 2 of the `rc.boot` script is complete, the boot process switches from the RAM file system to the file systems that are stored on the hard disk.
3. Then the **init** process runs the processes defined by records in the `/etc/inittab` file. One of the instructions in the `/etc/inittab` file runs phase 3 of the `rc.boot` script, which includes the following steps:

   a) Mount the `/tmp` hard disk file system.

   b) Start the configuration manager phase 2 to configure all remaining devices.

   c) Use the **savebase** command to save the customized data to the boot logical volume.

   d) Exit the `rc.boot` script.

At the end of this process, the system is up and ready for use.

Maintenance boot process

Occasions might arise when a boot is needed to perform special tasks such as installing new or updated software, performing diagnostic checks, or for maintenance. In this case, the system starts from a bootable medium such as a CD-ROM, DVD, tape drive, network, or disk drive.

The maintenance boot sequence of events is similar to the sequence of a normal boot.

1. The firmware performs basic testing on the system resources that are required for starting the operating system.
2. The firmware checks the user boot list. You can use the **bootlist** command to change the user boot list to suit your requirements. If the user boot list in non-volatile random access memory (NVRAM) is not valid or if no valid boot device is found, the default boot list is checked. In either case, the first valid boot device found in the boot list is used for starting the system.

   Note: For a normal boot, the operating system also maintains a default boot list and a user boot list, which are stored in NVRAM. Separate default boot list and user boot list are also maintained for starting the system in maintenance mode.
3. When a valid boot device is found, the first record or program sector number (PSN) is checked. If it is a valid boot record, it is read into memory and is added to the initial program load (IPL) control block in memory. Included in the key boot record data are the starting location of the boot image on the boot device, the length of the boot image, and the offset to the entry point to start running when the boot image is in memory.
4. The boot image is read sequentially from the boot device into memory, starting at the location specified in NVRAM.
5. Control is passed to the kernel, which begins running programs in the RAM file system.
6. The ODM database contents determine which devices are present, and the **cfgmgr** command dynamically configures all devices found, including all disks which are to contain the root file system.
7. If a CD-ROM, DVD, tape, or the network is used to boot the system, the rootvg volume group (or rootvg) is not varied on, because the rootvg might not exist (as is the case when installing the operating system on a new system). Network configuration can occur at this time. No paging occurs when a maintenance boot is performed.

At the end of this process, the system is ready for installation, maintenance, or diagnostics.

Note: If the system is started from the hard disk, the rootvg is varied on, the hard disk root file system and the hard disk `/usr` file system are mounted in the RAM file system, a menu is displayed that allows you to enter various diagnostics modes or single-user mode. If you select single-user mode, you can continue the boot process and enter single-user mode, where the **init** run level is set to the letter S. The system is then ready for maintenance, software updates, or for running the **bosboot** command.

Related concepts

Boot process

There are three types of system boots and two resources that are required in order to boot the operating system.

RAM file system

The RAM file system, part of the boot image, is totally memory-resident and contains all programs that allow the boot process to continue. The files in the RAM file system are specific to the type of boot.

A maintenance boot RAM file system might not have the logical volume routines, because the rootvg might not need to be varied on. During a hard disk boot, however, it is desirable that the rootvg be varied on and paging activated as soon as possible. Although there are differences in these two boot scenarios, the structure of the RAM file system does not vary to a great extent.

The **init** command, which is located on the RAM file system, is a basic boot command interpreter program that is designed for use during the boot process. This boot command interpreter program controls the boot process by calling the `rc.boot` script. The `rc.boot` script determines from which device the machine was started. The boot device determines which devices must be configured on the RAM file system. If the machine is started over the network, the network devices need to be configured so that the client file systems can be remotely mounted. In the case of a tape, CD-ROM, or DVD boot, the console is configured to display the base operating system (BOS) installation menus. After the `rc.boot` script identifies the boot device, the appropriate configuration routines are called from the RAM file system. The `rc.boot` script is called twice by the boot command interpreter program to match the two configuration phases during the boot process. A third call to `rc.boot` occurs during a disk or a network boot when the real **init** command is called. The `inittab` file contains an `rc.boot` stanza that completes the final configuration of the machine.

The RAM file system for each boot device is also unique because of the various types of devices to be configured. A prototype file is associated with each type of boot device. The prototype file is a template of files making up the RAM file system. The **bosboot** command uses the **mkfs** command to create the RAM file system using the various prototype files. See the **bosboot** command for more details.

Related concepts

Boot process

There are three types of system boots and two resources that are required in order to boot the operating system.

Troubleshooting system startup

Use these troubleshooting methods to tackle some of the basic problems that might occur when your system is starting. If the troubleshooting information does not address your problem, contact your service representative.

Systems that will not boot

If a system will not boot from the hard disk, you may still be able to gain access to the system in order to ascertain and correct the problem.

If you have a system that will not boot from the hard disk, see the procedure on how to access a system that will not boot in Troubleshooting your installation in the *Installation and migration*.

This procedure enables you to get a system prompt so that you can attempt to recover data from the system or perform corrective action enabling the system to boot from the hard disk.

Note:

- This procedure is intended only for experienced system managers who have knowledge of how to boot or recover data from a system that is unable to boot from the hard disk. Most users should not attempt this procedure, but should contact their service representative.
- This procedure is not intended for system managers who have just completed a new installation, because in this case the system does not contain data that needs to be recovered. If you are unable to boot from the hard disk after completing a new installation, contact your service representative.

Related reference

Boot problem diagnostics

A variety of factors can cause a system to be unable to boot.

Boot problem diagnostics

A variety of factors can cause a system to be unable to boot.

Some of these factors are:

- Hardware problems
- Defective boot tapes or CD-ROMs
- Improperly configured network boot servers
- Damaged file systems
- Errors in scripts such as `/sbin/rc.boot`

If the boot process halts with reference code 2702 and displays the message "INSUFFICIENT ENTITLED MEMORY" use the HMC to increase the amount of entitled memory available for that partition.

Related concepts

Systems that will not boot

If a system will not boot from the hard disk, you may still be able to gain access to the system in order to ascertain and correct the problem.

System backup

Once your system is in use, your next consideration should be to back up the file systems, directories, and files. If you back up your file systems, you can restore files or file systems in the event of a hard disk crash. There are different methods for backing up information.

Backing up file systems, directories, and files represents a significant investment of time and effort. At the same time, all computer files are potentially easy to change or erase, either intentionally or by accident.

Attention: When a hard disk crashes, the information contained on that disk is destroyed. The only way to recover the destroyed data is to retrieve the information from your backup copy.

If you use a careful and methodical approach to backing up your file systems, you should always be able to restore recent versions of files or file systems with little difficulty.

Several methods exist for backing up information. One of the most frequently used methods is called *backup by name*, *file name archive*, or *regular backup*. This is a copy of a file system, directory, or file that is kept for file transfer or in case the original data is unintentionally changed or destroyed. This method of backup is done when the **i** flag is specified and is used to make a backup copy of individual files and directories. It is a method commonly used by individual users to back up their accounts.

Another frequently used method is called *backup by i-node*, *file system archive*, or *archive backup*. This method of backup is done when the **i** flag is *not* specified. This is used for future reference, historical purposes, or for recovery if the original data is damaged or lost. It is used to make a backup copy of an entire file system and is the method commonly used by system administrators to back up large groups of files, such as all of the user accounts in `/home`. A file system backup allows incremental backups to be performed easily. An incremental backup backs up all files that have been modified since a specified previous backup.

The **compress** and **pack** commands enable you to compress files for storage, and the **uncompress** and **unpack** commands unpack the files once they have been restored. The process of packing and unpacking files takes time, but once packed, the data uses less space on the backup medium. For more information about these commands, see **compress**, **pack**, **uncompress**, and **unpack**.

Several commands create backups and archives. Because of this, data that has been backed up needs to be labeled as to which command was used to initiate the backup, and how the backup was made (by name or by file system).

Item	Description
backup	Backs up files by name or by file system. For more information, see **backup**.

Item	Description
mksysb	Creates an installable image of the rootvg. For more information, see **mksysb**.
cpio	Copies files into and out of archive storage. For more information, see **cpio**.
dd	Converts and copies a file. Commonly used to convert and copy data to and from systems running other operating systems, for example, mainframes. **dd** does not group multiple files into one archive; it is used to manipulate and move data. For more information, see **dd**.
tar	Creates or manipulates `tar` format archives. For more information, see **tar**.
rdump	Backs up files by file system onto a remote machine's device. For more information, see **rdump**.
pax	(POSIX-conformant archive utility) Reads and writes **tar** and **cpio** archives. For more information, see **pax**.

Related concepts

Backup for BSD 4.3 system managers
BSD 4.3 system managers can back up data.

Related tasks

Backing up user files or file systems
Two procedures can be used to back up files and file systems: the SMIT fast paths **smit backfile** or **smit backfilesys**, and the **backup** command.

Backup concepts

Before you start backing up your data, you need to understand the types of data, policies, and media that you can use.

Backup policies

No single backup policy can meet the needs of all users. A policy that works well for a system with one user, for example, could be inadequate for a system that serves one hundred users. Likewise, a policy developed for a system on which many files are changed daily would be inefficient for a system on which data changes infrequently.

Whatever the appropriate backup strategy for your site, it is very important that one exist and that backups be done frequently and regularly. It is difficult to recover from data loss if a good backup strategy has not been implemented.

Only you can determine the best backup policy for your system, but the following general guidelines might be helpful:

- **Make sure you can recover from major losses.**

  Can your system continue to run after any single fixed disk failure? Can you recover your system if all the fixed disks should fail? Could you recover your system if you lost your backup diskettes or tape to fire or theft? If data were lost, how difficult would it be to re-create it? Think through possible, even unlikely, major losses, and design a backup policy that enables you to recover your system after any of them.
- **Check your backups periodically.**

  Backup media and their hardware can be unreliable. A large library of backup tapes or diskettes is useless if data cannot be read back onto a fixed disk. To make certain that your backups are usable, display the table of contents from the backup tape periodically (using **restore -T** or **tar -t** for archive tapes). If you use diskettes for your backups and have more than one diskette drive, read diskettes from a drive other than the one on which they were created. You might want the security of repeating each level 0 backup with a second set of media. If you use a streaming tape device for backups, you can use the **tapechk** command to perform rudimentary consistency checks on the tape. For more information about these commands, see **restore -T**, **tar -t**, and **tapechk**.
- **Keep old backups.**

Develop a regular cycle for reusing your backup media; however, do not reuse all of your backup media. Sometimes it is months before you or some other user of your system notices that an important file is damaged or missing. Save old backups for such possibilities. For example, you could have the following three cycles of backup tapes or diskettes:

- Once per week, recycle all daily diskettes except the Friday diskette.
- Once per month, recycle all Friday diskettes except for the one from the last Friday of the month. This makes the last four Friday backups always available.
- Once per quarter, recycle all monthly diskettes except for the last one. Keep the last monthly diskette from each quarter indefinitely, preferably in a different building.

- **Check file systems before backing up.**

  A backup made from a damaged file system might be useless. Before making your backups, it is good policy to check the integrity of the file system with the **fsck** command. For more information, see **fsck**.

- **Ensure files are not in use during a backup.**

  Do not use the system when you make your backups. If the system is in use, files can change while they are being backed up, and the backup copy will not be accurate.

- **Back up your system before major changes are made to the system.**

  It is always good policy to back up your entire system before any hardware testing or repair work is performed or before you install any new devices, programs, or other system features.

- **Other factors.**

  When planning and implementing a backup strategy, consider the following factors:

  - How often does the data change? The operating system data does not change very often, so you do not need to back it up frequently. User data, on the other hand, usually changes frequently, so you should back it up frequently.
  - How many users are on the system? The number of users affects the amount of storage media and frequency required for backups.
  - How difficult would it be to re-create the data? It is important to consider that some data cannot be re-created if there is no backup available.

Having a backup strategy in place to preserve your data is very important. Evaluating the needs of your site will help you to determine the backup policy that is best for you. Perform user information backups frequently and regularly. Recovering from data loss is very difficult if a good backup strategy has not been implemented.

Note: For the backup of named pipes (FIFO special files) the pipes can be either closed or open. However, the restoration fails when the backup is done on open named pipes. When restoring a FIFO special file, its i-node is all that is required to recreate it because it contains all its characteristic information. The content of the named pipe is not relevant for restoration. Therefore, the file size during backup is zero (all the FIFOs closed) before the backup is made.

Attention: System backup and restoration procedures require that the system be restored on the same type of platform from which the backup was made. In particular, the CPU and I/O planar boards must be of the same type.

Backup media

Several different types of backup media are available. The different types of backup media available to your specific system configuration depend upon both your software and hardware.

Several types of backup media are available. The types of backup media available to your specific system configuration depend upon your software and hardware. The types most frequently used are tapes (8-mm tape and 9-track tape), diskettes (5.25-inch diskette and 3.5-inch diskette), remote archives, and alternate local hard disks. Unless you specify a different device using the **backup -f** command, the **backup** command automatically writes its output to `/dev/rfd0`, which is the diskette drive.

Attention: Running the **backup** command results in the loss of all material previously stored on the selected backup medium.

Diskettes

Diskettes are the standard backup medium. Unless you specify a different device using the **backup -f** command, the **backup** command automatically writes its output to the `/dev/rfd0` device, which is the diskette drive. To back up data to the default tape device, type `/dev/rmt0` and press Enter.

Be careful when you handle diskettes. Because each piece of information occupies such a small area on the diskette, small scratches, dust, food, or tobacco particles can make the information unusable. Be sure to remember the following:

- Do not touch the recording surfaces.
- Keep diskettes away from magnets and magnetic field sources, such as telephones, dictation equipment, and electronic calculators.
- Keep diskettes away from extreme heat and cold. The recommended temperature range is 10 degrees Celsius to 60 degrees Celsius (50 degrees Fahrenheit to 140 degrees Fahrenheit).
- Proper care helps prevent loss of information.
- Make backup copies of your diskettes regularly.

Attention: Diskette drives and diskettes must be the correct type to store data successfully. If you use the wrong diskette in your 3.5-inch diskette drive, the data on the diskette could be destroyed.

The diskette drive uses the following 3.5-inch diskettes:

- 1 MB capacity (stores approximately 720 KB of data)
- 2 MB capacity (stores approximately 1.44 MB of data)

Tapes

Because of their high capacity and durability, tapes are often chosen for storing large files or many files, such as archive copies of file systems. They are also used for transferring many files from one system to another. Tapes are not widely used for storing frequently accessed files because other media provide much faster access times.

Tape files are created using commands such as **backup**, **cpio**, and **tar**, which open a tape drive, write to it, and close it.

Backup strategy

There are two methods of backing up large amounts of data.

- Complete system backup
- Incremental backup

To understand these two types of backups and which one is right for a site or system, it is important to have an understanding of file system structure and data placement. After you have decided on a strategy for data placement, you can develop a backup strategy for that data.

Related tasks

Implementing scheduled backups
This procedure describes how to develop and use a script to perform a weekly full backup and daily incremental backups of user files.

System data versus user data
Data is defined as programs or text and for this discussion is broken down into two classes:

- System data, which makes up the operating system and its extensions. This data is always to be kept in the system file systems, namely / (root), `/usr`, `/tmp`, `/var`, and so on.
- User data is typically local data that individuals need to complete their specific tasks. This data is to be kept in the `/home` file system or in file systems that are created specifically for user data.

User programs and text are not to be placed in file systems designed to contain system data. For example, a system manager might create a new file system and mount it over `/local`. An exception is `/tmp`, which is used for temporary storage of system and user data.

Backups
In general, backups of user and system data are kept in case the data is accidentally removed or if there is a disk failure. It is easier to manage backups when user data is kept separate from system data.

The following are reasons for keeping system data separate from user data:

- User data tends to change much more often than operating system data. Backup images are much smaller if the system data is not backed up into the same image as the user data. The number of users affects the storage media and frequency that is required for backup.
- It is quicker and easier to restore user data when it is kept separate. Restoring the operating system along with the user data requires extra time and effort. The reason is that the method that is used to recover the operating system data involves starting the system from removable media (tape or CD) and installing the system backup.

To back up the system data, unmount all user file systems, including `/home` with the **umount** command. If these file systems are in use, you cannot unmount them. Schedule the backups at low usage times so they can be unmounted; if the user data file systems remain mounted, they are backed up along with the operating system data. Use the **mount** command to ensure that only the operating system file systems are mounted.

The only mounted file systems are `/`, `/usr`, `/var`, and `/tmp`, and the result of the **mount** command can be similar to the following output:

```
node  mounted  mounted over vfs       date         options
      /dev/hd4     /        jfs   Jun 11 10:36  rw,log=/dev/hd8
      /dev/hd2     /usr     jfs   Jun 11 10:36  rw,log=/dev/hd8
      /dev/hd9var  /var     jfs   Jun 11 10:36  rw,log=/dev/hd8
      /dev/hd      /tmp     jfs   Jun 11 10:36  rw,log=/dev/hd8
```

After you are certain that all user file systems are unmounted, you are now ready to back up the operating system data.

When you finish backing up the operating system, mount the user file system by using the **smit mount** command. Next, you can back up files, file systems, or other volume groups, depending on your needs.

Related concepts
System image and user-defined volume groups backup
The rootvg is stored on a hard disk, or group of disks, and contains start up files, the BOS, configuration information, and any optional software products. A *user-defined volume group* (also called *nonrootvg volume group*) typically contains data files and application software.

System replication (cloning)
Cloning saves configuration data along with user or system data. For example, you might want to replicate a system or volume group; it is sometimes called cloning.

You can then install this image onto another system and can use it just like the first system. The **mksysb** command is used to clone the rootvg volume group, which contains the operating system, while the **savevg** command is used to clone a volume group.

Command summary for backup files and storage media
Commands are available for backing up files and storing data.

Item	Description
backup	Backs up files and file systems
compress	Compresses and expands data

Item	Description
cpio	Copies files into and out of archive storage and directories
fdformat	Formats diskettes
flcopy	Copies to and from diskettes
format	Formats diskettes
fsck	Checks file system consistency and interactively repairs the file system
pack	Compresses files
restore	Copies previously backed-up file systems or files, which were created by the **backup** command from a local device
tapechk	Checks consistency of the streaming tape device
tar	Manipulates archives
tcopy	Copies a magnetic tape
uncompress	Compresses and expands data
unpack	Expands files

Administering system backups

There are multiple ways to backup your system and restore a system backup.

Backing up user files or file systems

Two procedures can be used to back up files and file systems: the SMIT fast paths **smit backfile** or **smit backfilesys**, and the **backup** command.

- If you are backing up by i-node file systems that may be in use, unmount them first to prevent inconsistencies.

  Attention: If you attempt to back up a mounted file system, a warning message is displayed. The **backup** command continues, but inconsistencies in the file system may occur. This warning does not apply to the root (/) file system.

- To prevent errors, make sure the backup device has been cleaned recently.

To back up user files and file systems, you can use the SMIT fast paths **smit backfile** or **smit backfilesys**.

You can use the SMIT interface for backing up single and small file systems by name, such as /home on your local system. Note that SMIT cannot make archives in any other format than that provided by the **backup** command. Also, not every flag of the **backup** command is available through SMIT. SMIT might hang if multiple tapes or disks are needed during the backup.

Use the **backup** command when you want to back up large and multiple file systems. You can specify a level number to control how much data is backed up (full, 0; incremental, 1-9). Using the **backup** command is the only way you can specify the level number on backups.

The **backup** command creates copies in one of the two following backup formats:

- Specific files backed up by name using the **-i** flag.
- Entire file systems backed up by i-node using the **-Level** and **FileSystem** parameters. The file system is defragmented when it is restored from backup.

  Attention: Backing up by i-node does not work correctly for files that have a user ID (UID) or a group ID (GID) greater than 65535. These files are backed up with UID or GID truncated and will, therefore, have the wrong UID or GID attributes when restored. For these cases, you must back up by name.

Backing Up User Files or File Systems Tasks		
Task	***SMIT Fast Path***	***Command or File***
Back Up User Files	`smit backfile`	1. Log in to your user account. 2. Backup: `find . -print | backup -ivf /dev/rmt0`
Back Up User File Systems	`smit backfilesys`	1. Unmount files systems that you plan to back up. For example: `umount all` or `umount /home /filesys1` 2. Verify the file systems. For example: `fsck /home /filesys1` 3. Back up by i-node. For example: `backup -5 -uf/dev/rmt0 /home/libr` 4. Restore the files using the following command: **`restore -t`**

Note: If this command generates an error message, you must repeat the entire backup.

Related concepts

System backup

Once your system is in use, your next consideration should be to back up the file systems, directories, and files. If you back up your file systems, you can restore files or file systems in the event of a hard disk crash. There are different methods for backing up information.

Restoring backed-up files

After the data has been correctly backed up, there are several different methods of restoring the data based upon the type of backup command you used.

You need to know how your backup or archive was created to restore it correctly. Each backup procedure gives information about restoring data. For example, if you use the **backup** command, you can specify a backup either by file system or by name. That backup must be restored the way it was done, by file system or by name.

Several commands restore backed up data, such as:

Item	Description
restore	Copies files created by the **backup** command. For more information about using this command, see the "Restoring files using the **restore**command' section below.
rrestore	Copies file systems backed up on a remote machine to the local machine.
cpio	Copies files into and out of archive storage.
tar	Creates or manipulates **tar** archives.
pax	(POSIX-conformant archive utility) Reads and writes **tar** and **cpio** archives.

The following sections discuss the **restore** and **smit** commands.

Note:

- Files must be restored using the same method by which they were backed up. For example, if a file system was backed up by name, it must be restored by name.

- When more than one diskette is required, the **restore** command reads the diskette that is mounted, prompts you for a new one, and waits for your response. After inserting the new diskette, press the Enter key to continue restoring files.

Restoring files using the `restore` command

Use the **restore** command to read files written by the **backup** command and restore them on your local system.

See the following examples:

- To list the names of files previously backed up, type the following:

```
restore -T
```

Information is read from the `/dev/rfd0` default backup device. If individual files are backed up, only the file names are displayed. If an entire file system is backed up, the i-node number is also shown.

- To restore files to the main file system, type the following:

```
restore -x -v
```

The **-x** flag extracts all the files from the backup media and restores them to their proper places in the file system. The **-v** flag displays a progress report as each file is restored. If a file system backup is being restored, the files are named with their i-node numbers. Otherwise, only the names are displayed.

- To copy the `/home/mike/manual/chap1` file, type the following:

```
restore -xv /home/mike/manual/chap1
```

This command extracts the `/home/mike/manual/chap1` file from the backup medium and restores it. The `/home/mike/manual/chap1` file must be a name that the **restore -T** command can display.

- To copy all the files in a directory named `manual`, type the following:

```
restore -xdv manual
```

This command restores the `manual` directory and the files in it. If the directory does not exist, a directory named `manual` is created in the current directory to hold the files being restored.

Restoring files using the `smit` command

Use the **smit** command to run the **restore** command, which reads files written by the **backup** command and restores them on your local system.

1. At the prompt, type the following:

```
smit restore
```

2. Make your entry in the **Target DIRECTORY** field. This is the directory where you want the restored files to reside.
3. Proceed to the **BACKUP device** or **FILE** field and enter the output device name, as in the following example for a raw magnetic tape device:

```
/dev/rmt0
```

If the device is not available, a message similar to the following is displayed:

```
Cannot open /dev/rmtX, no such file or directory.
```

This message indicates that the system cannot reach the device driver because there is no file for **rmtX** in the `/dev` directory. Only items in the `available` state are in the `/dev` directory.

4. For the **NUMBER of blocks to read in a single input** field, the default is recommended.

5. Press Enter to restore the specified file system or directory.

Creating a remote archive

Use this procedure to archive files to a remote tape device.

Running AIX systems cannot mount a remote tape device as if it were local to the system; however, data can be sent to a remote machine tape device using the **rsh** command. The following procedure writes to a single tape only. Multiple-tape archives require specialized application software.

In the following procedure, assume the following:

blocksize
Represents the target tape device blocksize.

remotehost
Is the name of the target system (the system that has the tape drive).

sourcehost
Is the name of the source system (the system being archived).

/dev/rmt0
Is the name of the remote tape device

pathname
Represents the full pathname of a required directory or file.

When using the following instructions, assume that both the local and remote user is root.

1. Ensure you have access to the remote machine.

   The source machine must have access to the system with the tape drive. (The target system can be accessed using any of the defined users on that system, but the user name must have root authority to do many of the following steps.)
2. Using your favorite editor, create a file in the / (root) directory of the target system called `.rhosts` that allows the source system access to the target system.

   You need to add the authorized host name and user ID to this file. To determine the name of the source machine for the `.rhosts` file, you can use the following command:

   ```
   host SourceIPaddress
   ```

   For the purposes of this example, assume you add the following line to the `.rhosts` file:

   ```
   sourcehost.mynet.com   root
   ```
3. Save the file and then change its permissions using the following command:

   ```
   chmod 600 .rhosts
   ```
4. Use the **rsh** command to test your access from the source machine. For example:

   ```
   rsh remotehost
   ```

   If everything is set up correctly, you should be granted shell access to the remote machine. You should not see a login prompt asking for a user name. Type `exit` to log out of this test shell.
5. Decide on the appropriate tape device blocksize.

   The following are the recommended values:

Item	Description
9-track or 0.25-in. media blocksize:	512
8-mm or 4-mm media blocksize:	1024

If you are unsure and want to check the current block size of the tape device, use the **tctl** command. For example:

```
tctl -f /dev/rmt0 status
```

If you want to change the tape blocksize, use the **chdev** command. For example:

```
chdev -l rmt0 -a block_size=1024
```

6. Create your archive using one of the following methods:

   Backup by Name
   To remotely create a backup archive by name, use the following command:

   ```
   find pathname -print | backup -ivqf- | rsh remotehost \
       "dd of=/dev/rmt0 bs=blocksize conv=sync"
   ```

   Backup by inode
   To remotely create a backup archive by inode, first unmount your file system then use the **backup** command. For example:

   ```
   umount /myfs
   backup -0 -uf- /myfs | rsh remotehost \
        "dd of=/dev/rmt0 bs=blocksize conv=sync"
   ```

   Create and Copy an Archive to Remote Tape
   To create and copy an archive to the remote tape device, use the following command:

   ```
   find pathname -print | cpio -ovcB | rsh remotehost \
       "dd ibs=5120 obs=blocksize of=/dev/rmt0"
   ```

   Create a tar Archive
   To remotely create a **tar** archive, use the following command:

   ```
   tar -cvdf- pathname | rsh remotehost \
       "dd of=/dev/rmt0 bs=blocksize conv=sync"
   ```

   Create a Remote Dump
   To remotely create a remote dump of the `/myfs` file system, use the **rdump** command:

   ```
   rdump -u -0 -f remotehost:/dev/rmt0 /myfs
   ```

   The **-u** flag tells the system to update the current backup level records in the `/etc/dumpdates` file. The **-0** is the setting of the *Level* flag. Backup level 0 specifies that all the files in the `/myfs` directory are to be backed up.

7. Restore your remote archive using one of the following methods:

   Restore a Backup by Name
   To restore a remote backup archive by name, use the following command:

   ```
   rsh remotehost "dd if=/dev/rmt0 bs=blocksize" | restore \
        -xvqdf- pathname
   ```

   Restore a Backup by inode
   To restore a remote backup archive by inode, use the following command:

   ```
   rsh remotehost "dd if=/dev/rmt0 bs=blocksize" | restore \
       -xvqf- pathname
   ```

   Restore a Remote cpio Archive
   To restore a remote archive created with the **cpio** command, use the following command:

   ```
   rsh remotehost "dd if=/dev/rmt0 ibs=blocksize obs=5120" | \
        cpio -icvdumB
   ```

Restore a tar Archive

To restore a remote **tar** archive, use the following command:

```
rsh remotehost "dd if=/dev/rmt0 bs=blocksize" | tar -xvpf- pathname
```

Restore a Remote Dump

To restore a remote dump of the /myfs file system, use the following command:

```
cd /myfs
rrestore -rvf remotehost:/dev/rmt0
```

Restoring user files from a backup image

If you need to restore a backup image destroyed by accident, your most difficult problem is determining which of the backup tapes contains this file. The **restore -T** command can be used to list the contents of an archive. It is a good idea to restore the file in the /tmp directory so that you do not accidentally overwrite the user's other files.

Make sure the device is connected and available. To check availability, type:

```
lsdev -C | pg
```

If the backup strategy included incremental backups, then it is helpful to find out from the user when the file was most recently modified. This helps to determine which incremental backup contains the file. If this information cannot be obtained or is found to be incorrect, then start searching the incremental backups in reverse order (7, 6, 5, ...). For incremental file system backups, the **-i** flag (interactive mode) of the **restore** command is very useful in both locating and restoring the lost file. (Interactive mode is also useful for restoring an individual user's account from a backup of the /home file system.)

The procedures in the following table describe how to implement a level 0 (full) restoration of a directory or file system.

Restoring from Backup Image Tasks		
Task	*SMIT Fast Path*	*Command or File*
Restore Individual User Files	**smit restfile**	See **restore** command.
Restoring a User File System	**smit restfilesys**	1. **mkfs /dev/hd1** 2. **mount /dev/hd1 /filesys** 3. **cd /filesys** 4. **restore -r**
Restoring a User Volume Group	**smit restvg**	See **restvg -q** command.

Restoring access to an unlinked or deleted system library

When the existing **libc.a** library is not available, most operating system commands are not recognized.

The most likely causes for this type of problem are the following:

- The link in /usr/lib no longer exists.
- The file in /usr/ccs/lib has been deleted.

The following procedure describes how to restore access to the **libc.a** library. This procedure requires system downtime. If possible, schedule your downtime when it least impacts your workload to protect yourself from a possible loss of data or functionality.

The information in this how-to scenario was tested using specific versions of AIX. The results you obtain might vary significantly depending on your version and level of AIX.

Related information

mount command

unmount command
reboot command

Restoring a deleted symbolic link

Use the following procedure to restore a symbolic link from the `/usr/lib/libc.a` library to the `/usr/ccs/lib/libc.a` path.

The information in this how-to scenario was tested using specific versions of AIX. The results you obtain might vary significantly depending on your version and level of AIX.

1. With root authority, set the **LIBPATH** environment variable to point to the `/usr/ccs/lib` directory by typing the following commands:

   ```
   # LIBPATH=/usr/ccs/lib:/usr/lib
   # export LIBPATH
   ```

   At this point, you should be able to execute system commands.
2. To restore the links from the `/usr/lib/libc.a` library and the `/lib` directory to the `/usr/lib` directory, type the following commands:

   ```
   ln -s /usr/ccs/lib/libc.a /usr/lib/libc.a
   ln -s /usr/lib /lib
   ```

   At this point, commands should run as before. If you still do not have access to a shell, skip the rest of this procedure and continue with the next section, “Restoring a deleted system library file” on page 30.
3. Type the following command to unset the LIBPATH environment variable.

   ```
   unset LIBPATH
   ```

Restoring a deleted system library file

This procedure to restore a deleted system library file requires system downtime. The system is booted and then the library is restored from a recent **mksysb** tape.

1. Before your reboot, ensure the **PROMPT** field in the `bosinst.data` file is set to `yes`.
2. Insert a recent **mksysb** tape into the tape drive.

   The **mksysb** *must* contain the same OS and maintenance package or technology level as the installed system. If you restore a `libc.a` library from a **mksysb** that conflicts with the level on the installed system, you will not be able to issue commands.
3. Reboot the machine.
4. When the screen of icons appears, or when you hear a double beep, press the F1 key repeatedly until the System Management Services menu is displayed.
5. Select **Multiboot**.
6. Select **Install From**.
7. Select the tape device that holds the **mksysb** and then select **Install**.

   It can take several minutes before the next prompt appears.
8. Define your current system as the system console by pressing the F1 key and press Enter.
9. Select the number of your preferred language and press Enter.
10. Select **Start Maintenance Mode for System Recovery** by typing 3 and press Enter.
11. Select **Access a Root Volume Group**. A message displays explaining that you will not be able to return to the Installation menus without rebooting if you change the root volume group at this point.
12. Type `0` and press Enter.
13. Type the number of the appropriate volume group from the list and press Enter.
14. Select **Access this Volume Group** by typing 2 and press Enter.
15. Mount the / (root) and `/usr` file systems by typing the following commands:

```
mount /dev/hd4 /mnt
mount /dev/hd2 /mnt/usr
cd /mnt
```

16. To restore the symbolic link for the `libc.a` library, if needed, type the following command:

    ```
    ln -s /usr/ccs/lib/libc.a /mnt/usr/lib/libc.a
    ```

    After the command runs, do one of the following:

    - If the command is successful, skip to step 20.
    - If a message displays that the link already exists, continue with step 17.

17. Set the block size of the tape drive by issuing the following commands, where *X* is the number of the appropriate tape drive.

    ```
    tctl -f /dev/rmtX rewind
    tctl -f /dev/rmtX.1 fsf 1
    restbyname -xvqf /dev/rmtX.1 ./tapeblksz
    cat tapeblksz
    ```

    If the value from the **cat tapeblksz** command is *not equal* to 512, type the following commands, replacing *Y* with the value from the **cat tapeblksz** command:

    ```
    ln -sf /mnt/usr/lib/methods /etc/methods
    /etc/methods/chgdevn -l rmtX -a block_size=Y
    ```

    You should receive a message that rmt*X* has been changed.

18. Ensure the tape is at the correct location for restoring the library by typing the following commands (where *X* is the number of the appropriate tape drive):

    ```
    tctl -f /dev/rmtX rewind
    tctl -f /dev/rmtX.1 fsf 3
    ```

19. Restore the missing library using one of the following commands (where *X* is the number of the appropriate tape drive):

    - To restore the `libc.a` library only, type the following command:

      ```
      restbyname -xvqf /dev/rmtX.1 ./usr/ccs/lib/libc.a
      ```

    - To restore the `/usr/ccs/lib` directory, type the following command:

      ```
      restbyname -xvqf /dev/rmtX.1 ./usr/ccs/lib
      ```

    - To restore the `/usr/ccs/bin` directory, type the following command:

      ```
      restbyname -xvqf /dev/rmtX.1 ./usr/ccs/bin
      ```

20. Flush the data to disk by typing the following commands:

    ```
    cd /mnt/usr/sbin
    ./sync;./sync;./sync
    ```

21. Unmount the `/usr` and `/` (root) file systems by typing the following commands:

    ```
    cd /
    umount /dev/hd2
    umount /dev/hd4
    ```

    If either **umount** command fails, cycle power on this machine and begin this procedure again.

22. Reboot the system by typing the following command:

    ```
    reboot
    ```

After the system is rebooted, operating system commands should be available.

Recreating a corrupted boot image

The following procedure describes how to identify a corrupted boot image and re-create it.

If your machine is currently running and you know the boot image has been corrupted or deleted, recreate the boot image by running the **bosboot** command with root authority.

Attention: Never reboot the system when you suspect the boot image is corrupted.

The following procedure assumes your system is not rebooting correctly because of a corrupted boot image. If possible, protect your system from a possible loss of data or functionality by scheduling your downtime when it least impacts your workload.

The information in this how-to scenario was tested using specific versions of AIX. The results you obtain might vary significantly depending on your version and level of AIX.

1. Insert the product media into the appropriate drive.
2. Power on the machine following the instructions provided with your system.
3. From the **System Management Services** menu, select **Multiboot**.
4. From the next screen, select **Install From**.
5. Select the device that holds the product media and then select **Install**.
6. Select the AIX version icon.
7. Follow the online instructions until you can select which mode you use for installation. At that point, select **Start Maintenance Mode for System Recovery**.
8. Select **Access a Root Volume Group**.
9. Follow the online instructions until you can select **Access this Volume Group and start a shell**.
10. Use the **bosboot** command to re-create the boot image. For example:

    ```
    bosboot -a -d /dev/hdisk0
    ```

    If the command fails and you receive the following message:

    ```
    0301-165 bosboot: WARNING! bosboot failed - do not attempt to boot device.
    ```

    Try to resolve the problem using one of the following options, and then run the **bosboot** command again until you have successfully created a boot image:

    - Delete the default boot logical volume (hd5) and then create a new hd5.

    Or

    - Run diagnostics on the hard disk. Repair or replace, as necessary.

    If the **bosboot** command continues to fail, contact your customer support representative.

    Attention: If the **bosboot** command fails while creating a boot image, do not reboot your machine.
11. When the **bosboot** command is successful, use the **reboot** command to reboot your system.

Related concepts

System startup

When the base operating system starts, the system initiates a complex set of tasks. Under normal conditions, these tasks are performed automatically.

Related information

bosboot command

Making an online backup of a JFS

Making an online backup of a mounted journaled file system (JFS) or enhanced journaled file system (JFS2) creates a static image of the logical volume that contains the file system.

To make an online backup of a mounted JFS, the logical volume that the file system resides on and the logical volume that its log resides on must be mirrored.

Note: Because the file writes are asynchronous, the split-off copy might not contain all data that was written immediately before the split. Any modifications that begin after the split begins might not be present in the backup copy. Therefore, it is recommended that file system activity be minimal while the split is taking place.

The information in this how-to scenario was tested using specific versions of AIX. The results you obtain might vary significantly depending on your version and level of AIX.

To split off a mirrored copy of the `/home/xyz` file system to a new mount point named `/jfsstaticcopy`, type the following:

```
chfs -a splitcopy=/jfsstaticcopy /home/xyz
```

You can control which mirrored copy is used as the backup by using the **copy** attribute. The second mirrored copy is the default if a copy is not specified by the user. For example:

```
chfs -a splitcopy=/jfsstaticcopy -a copy=1 /home/xyz
```

At this point, a read-only copy of the file system is available in `/jfsstaticcopy`. Any changes made to the original file system after the copy is split off are not reflected in the backup copy.

To reintegrate the JFS split image as a mirrored copy at the `/testcopy` mount point, use the following command:

```
rmfs /testcopy
```

The **rmfs** command removes the file system copy from its split-off state and allows it to be reintegrated as a mirrored copy.

Making and backing up a snapshot of a JFS2

You can make a snapshot of a mounted JFS2 that establishes a consistent block-level image of the file system at a point in time.

The information in this how-to scenario was tested using specific versions of AIX. The results you obtain might vary significantly depending on your version and level of AIX.

The snapshot image remains stable even as the file system that was used to create the snapshot, called the *snappedFS*, continues to change. The snapshot retains the same security permissions as the *snappedFS* had when the snapshot was made.

In the following scenario, you create a snapshot and back up the snapshot to removable media without unmounting or quiescing the file system, all with one command: **backsnap**. You can also use the snapshot for other purposes, such as accessing the files or directories as they existed when the snapshot was taken. You can do the various snapshot procedures by using SMIT or the **backsnap** and **snapshot** commands.

To create a snapshot of the `/home/abc/test` file system and back it up (by name) to the tape device `/dev/rmt0`, use the following command:

```
backsnap -m /tmp/snapshot -s size=16M -i f/dev/rmt0 /home/abc/test
```

This command creates a logical volume of 16 megabytes for the snapshot of the JFS2 file system (`/home/abc/test`). The snapshot is mounted on `/tmp/snapshot` and then a backup by name of the snapshot is made to the tape device. After the backup completes, the snapshot remains mounted. Use the **-R** flag with the **backsnap** command if you want the snapshot removed when the backup completes.

Related information
File Systems
backsnap command
chfs command
rmfs command
snapshot command

Making and backing up an external snapshot of a JFS2

You can make a snapshot of a mounted JFS2 that establishes a consistent block-level image of the file system at a point in time.

The snapshot image remains stable even as the file system that was used to create the snapshot, called the *snappedFS*, continues to change. The snapshot retains the same security permissions as the *snappedFS* had when the snapshot was made.

In the following scenario, you use the **backsnap** command to create an external snapshot and back up the snapshot to removable media without un mounting or quiescing the file system. You can also use the snapshot for other purposes, such as accessing the files or directories as they existed when the snapshot was taken. You can do the various snapshot procedures using SMIT, or the **backsnap** and **snapshot** commands.

To create an external snapshot of the `/home/abc/test` file system and back it up, by name, to the `/dev/rmt0` tape device, run the following command:

```
backsnap -m /tmp/snapshot -s size=16M -if/dev/rmt0 /home/abc/test
```

The previous command creates a logical volume of 16 MB for the snapshot of the `/home/abc/test` JFS2 file system. The snapshot is mounted on the `/tmp/snapshot` directory and then a backup of the snapshot, by name, is made to the tape device. After the backup is complete, the snapshot is unmounted but remains available. Use the `-R` flag with the **backsnap** command if you want the snapshot removed when the backup is completed.

Related information
File Systems

Making and backing up an internal snapshot of a JFS2

You can make a snapshot of a mounted JFS2 that establishes a consistent block-level image of the file system at a point in time.

The snapshot image remains stable even as the file system that was used to create the snapshot, called the *snappedFS*, continues to change. The snapshot retains the same security permissions as the *snappedFS* had when the snapshot was made.

In the following scenario, you use the **backsnap** command to create an internal snapshot and back up the snapshot to removable media without un mounting or quiescing the file system. You can also use the snapshot for other purposes, such as accessing the files or directories as they existed when the snapshot was taken. You can do the various snapshot procedures using SMIT, or the **backsnap** and **snapshot** commands.

To create an internal snapshot of the `/home/abc/test` file system and back it up, by name, to the `/dev/rmt0` tape device , run the following command:

```
backsnap -n mysnapshot -if/dev/rmt0 /home/abc/test
```

The previous command creates an internal snapshot, named `mysnapshot`, of the `/home/abc/test` file system. The snapshot is accessed from the `/home/abc/test/.snapshot/mysnapshot` directory and then a backup is made to the tape device. Use the `-R` flag with the **backsnap** command if you want the snapshot removed after the backup is completed.

Related information
File Systems

Compressing files (compress and pack commands)

Use the **compress** command and the **pack** command to compress files for storage.

Use the **uncompress** command and the **unpack** command to expand the restored files.

The process of compressing and expanding files takes time; however, after the files are packed, the data uses less space on the backup medium.

To compress a file system, use one of the following methods:

- Use the **-p** flag with the **backup** command.
- Use the **compress** or **pack** commands.

Advantages for compressing files include:

- Saving money and time by compressing files before sending them over a network.
- Saving storage and archiving system resources:
  - Compress file systems before making backups to preserve tape space.
  - Compress log files created by shell scripts that run at night; it is easy to have the script compress the file before it exits.
  - Compress files that are not currently being accessed. For example, the files belonging to a user who is away for extended leave can be compressed and placed into a **tar** archive on disk or to a tape and later be restored.

Note:

- The **compress** command might run out of working space in the file system while compressing. The command creates the compressed files before it deletes any of the uncompressed files, so it needs a space about 50% larger than the total size of the files.
- A file might fail to compress because it is already compressed. If the **compress** command cannot reduce file sizes, the command fails.

See the **compress** command for details about the return values but, in general, the problems encountered when compressing files can be summarized as follows:

- The command might run out of working space in the file system while compressing. Because the **compress** command creates the compressed files before it deletes any of the uncompressed files, it needs extra space-from 50% to 100% of the size of any given file.
- A file might fail to compress because it is already compressed. If the **compress** command cannot reduce the file size, it fails.

Compressing files using the compress command

Use the **compress** command to reduce the size of files using adaptive Lempel-Zev coding.

Each original file specified by the *File* parameter is replaced by a compressed file with a `.Z` appended to its name. The compressed file retains the same ownership, modes, and access and modification times of the original file. If no files are specified, the standard input is compressed to the standard output. If compression does not reduce the size of a file, a message is written to standard error and the original file is not replaced.

Use the **uncompress** command to restore compressed files to their original form.

The amount of compression depends on the size of the input, the number of bits per code specified by the *Bits* variable, and the distribution of common substrings. Typically, source code or English text is reduced by 50 to 60 percent. The compression of the **compress** command is generally more compact and takes less time to compute than the compression achieved by the **pack** command, which uses adaptive Huffman coding.

For example, to compress the `foo` file and write the percentage compression to standard error, type the following:

```
compress -v foo
```

Compressing files using the pack command
Use the **pack** command to store the file or files specified by the *File* parameter in a compressed form using Huffman coding.

The input file is replaced by a packed file with a name derived from the original file name (*File*`.z`), with the same access modes, access and modification dates, and owner as the original file. The input file name can contain no more than 253 bytes to allow space for the added `.z` suffix. If the **pack** command is successful, the original file is removed.

Use the **unpack** command to restore packed files to their original form.

If the **pack** command cannot create a smaller file, it stops processing and reports that it is unable to save space. (A failure to save space generally happens with small files or files with uniform character distribution.) The amount of space saved depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each .z file, you do not save space with files smaller than three blocks. Typically, text files are reduced 25 to 40 percent.

The exit value of the **pack** command is the number of files that it could not pack. Packing is not done under any of the following conditions:

- The file is already packed.
- The input file name has more than 253 bytes.
- The file has links.
- The file is a directory.
- The file cannot be opened.
- No storage blocks are saved by packing.
- A file named *File*`.z` already exists.
- The `.z` file cannot be created.
- An I/O error occurred during processing.

For example, to compress the files `chap1` and `chap2`, type the following:

```
pack chap1 chap2
```

This compresses `chap1` and `chap2` and replaces them with files named `chap1.z` and `chap2.z`. The **pack** command displays the percent decrease in size for each file.

Expanding compressed files (uncompress and unpack commands)
Use the **uncompress** and **unpack** commands to expand compressed files.

Expanding files using the uncompress command

Use the **uncompress** command to restore original files that were compressed by the **compress** command. Each compressed file specified by the *File* variable is removed and replaced by an expanded copy. The expanded file has the same name as the compressed version but without the `.Z` extension. The expanded file retains the same ownership, modes, and access and modification times as the original file. If no files are specified, standard input is expanded to standard output.

Although similar to the **uncompress** command, the **zcat** command always writes the expanded output to standard output.

For example, to uncompress the `foo` file, type the following:

```
uncompress foo
```

Expanding files using the unpack command

Use the **unpack** command to expand files created by the **pack** command. For each file specified, the **unpack** command searches for a file called *File*`.z`. If this file is a packed file, the **unpack** command replaces it with its expanded version. The **unpack** command renames the new file by removing the `.z`

suffix from *File*. The new file has the same access modes, access and modification dates, and owner as the original packed file.

The **unpack** command operates only on files ending in `.z`. As a result, when you specify a file name that does not end in `.z`, the **unpack** command adds the suffix and searches the directory for a file name with that suffix.

The exit value is the number of files that the **unpack** command was unable to unpack. A file cannot be unpacked if any of the following situations exists:

- The file name (exclusive of `.z`) has more than 253 bytes.
- The file cannot be opened.
- The file is not a packed file.
- A file with the unpacked file name already exists.
- The unpacked file cannot be created.

  Note: The **unpack** command writes a warning to standard error if the file it is unpacking has links. The new unpacked file has a different i-node (index node) number than the packed file from which it was created. However, any other files linked to the original i-node number of the packed file still exist and are still packed.

For example, to unpack the packed files `chap1.z` and `chap2.z`, type the following:

```
unpack chap1.z chap2
```

This expands the packed files `chap1.z` and `chap2.z`, and replaces them with files named `chap1` and `chap2`.

Note: You can provide the **unpack** command with file names with or without the `.z` suffix.

System image and user-defined volume groups backup

The rootvg is stored on a hard disk, or group of disks, and contains start up files, the BOS, configuration information, and any optional software products. A *user-defined volume group* (also called *nonrootvg volume group*) typically contains data files and application software.

You can backup an image of the system and volume groups using, SMIT, or command procedures. A backup image serves two purposes. One is to restore a corrupted system using the system backup image. The other is to transfer installed and configured software from one system to others.

The SMIT procedures use the **mksysb** command to create a backup image that can be stored either on tape or in a file. If you choose tape, the backup program writes a *boot image* to the tape, which makes it suitable for installing.

Note:

- Startup tapes cannot be made on or used to start a PowerPC-based personal computer.
- If you choose the SMIT method for backup, you must first install the `sysbr` fileset in the `bos.sysmgt` software package.

Related concepts

Backups

In general, backups of user and system data are kept in case the data is accidentally removed or if there is a disk failure. It is easier to manage backups when user data is kept separate from system data.

Backing up the system image and user-defined volume groups

You can make backups of the system image and the user-defined volume groups.

Before backing up the rootvg volume group:

- All hardware must already be installed, including external devices, such as tape and CD-ROM drives.

- This backup procedure requires the `sysbr` fileset, which is in the BOS System Management Tools and Applications software package. Type the following command to determine whether the `sysbr` fileset is installed on your system:

  ```
  lslpp -l bos.sysmgt.sysbr
  ```

  If your system has the `sysbr` fileset installed, continue the backup procedures.

  If the **lslpp** command does not list the `sysbr` fileset, install it before continuing with the backup procedure.

  ```
  installp -agqXd device bos.sysmgt.sysbr
  ```

  where `device` is the location of the software; for example, `/dev/rmt0` for a tape drive.

Before backing up a user-defined volume group:

- Before being saved, a volume group must be varied on and the file systems must be mounted.

  Attention: Executing the **savevg** command results in the loss of all material previously stored on the selected output medium.

- Make sure the backup device has been cleaned recently to prevent errors.

The following procedures describe how to make an installable image of your system.

Backing Up Your System Tasks		
Task	***SMIT Fast Path***	***Command or File***
Backing up the **rootvg** volume group	1. Log in as root. 2. Mount file systems for backup.[1] **smit mountfs** 3. Unmount any local directories that are mounted over another local directory. **smit umountfs** 4. Make at least 8.8MB of free disk space available in the `/tmp` directory.[2] 5. Back up: **smit mksysb** 6. Write-protect the backup media. 7. Record any backed-up root and user passwords.	1. Log in as root. 2. Mount file systems for backup.[1] See **mount** command. 3. Unmount any local directories that are mounted over another local directory. See **umount** command. 4. Make at least 8.8MB of free disk space available in the `/tmp` directory.[2] 5. Back up. See **mksysb** command. 6. Write-protect the backup media. 7. Record any backed-up root and user passwords.
Verify a Backup Tape[3]	**smit lsmksysb**	
Backing up a user-defined volume group[4]	**smit savevg**	1. Modify the file system size before backing up, if necessary.[5] **mkvgdata** *VGName* then edit `/tmp/vgdata/`*VGName*`/`*VGName*`.data` 2. Save the volume group. See the **savevg** command.

Note:

1. The **mksysb** command does not back up file systems mounted across an NFS network.

2. The **mksysb** command requires this working space for the duration of the backup. Use the **df** command, which reports in units of 512-byte blocks, to determine the free space in the `/tmp` directory. Use the **chfs** command to change the size of the file system, if necessary.
3. This procedure lists the contents of a **mksysb** backup tape. The contents list verifies most of the information on the tape but does not verify that the tape can be booted for installations. The only way to verify that the boot image on a **mksysb** tape functions correctly is by booting from the tape.
4. If you want to exclude files in a user-defined volume group from the backup image, create a file named `/etc/exclude.`*volume_group_name*, where *volume_group_name* is the name of the volume group that you want to back up. Then edit `/etc/exclude.`*volume_group_name* and enter the patterns of file names that you do not want included in your backup image. The patterns in this file are input to the pattern matching conventions of the **grep** command to determine which files are excluded from the backup.
5. If you choose to modify the *VGName*`.data` file to alter the size of a file system, you must not specify the **-i** flag or the **-m** flag with the **savevg** command, because the *VGName*`.data` file is overwritten.

Related information

Installing optional software products and service updates

Installing system backups

Pre-backup configuration

Configure the source system before creating a backup image of it. If, however, you plan to use a backup image for installing other, differently configured target systems, create the image *before* configuring the source system.

The *source* system is the system from which you created the backup copy. The *target* system is the system on which you are installing the backup copy.

The installation program automatically installs only the device support required for the hardware configuration of the installed machine. Therefore, if you are using a system backup to install other machines, you might need to install additional devices on the source system before making the backup image and using it to install one or more target systems.

Use the SMIT fast path, `smit devinst`, to install additional device support on the source system.

- If there is sufficient disk space on the source and target systems, install all device support.
- If there is limited disk space on the source and target systems, selectively install device support.

A backup transfers the following configurations from the source system to the target system:

- Paging space information
- Logical volume information
- rootvg information
- Placement of logical partitions (if you have selected the map option).

Related information

Installing optional software and service updates

Customizing your installation

File system mounts and unmounts

Before performing a backup, you must mount all file systems you want to back up and unmount all file systems you do not want to back up.

The Backup Methods procedure backs up only mounted file systems in the rootvg. You must, therefore, mount all file systems you want to back up before starting. Similarly, you must unmount file systems you do *not* want backed up.

This backup procedure backs up files twice if a local directory is mounted over another local directory in the same file system. For example, if you mount `/tmp` over `/usr/tmp`, the files in the `/tmp` directory

are backed up twice. This duplication might exceed the number of files a file system can hold, which can cause a future installation of the backup image to fail.

Security considerations for backups

If you install the backup image on other systems, you might not, for security reasons, want passwords and network addresses copied to the target systems.

Also, copying network addresses to a target system creates duplicate addresses that can disrupt network communications.

Backup image restoration

When installing the backup image, the system checks whether the target system has enough disk space to create all the logical volumes stored on the backup. If there is enough space, the entire backup is recovered. Otherwise, the installation halts and the system prompts you to choose more destination hard disks.

File systems created on the target system are the same size as they were on the source system, unless the SHRINK variable was set to `yes` in the `image.data` file before the backup image was made. An exception is the `/tmp` directory, which can be increased to allocate enough space for the **bosboot** command. For information about setting variables, see the `image.data` file.

When the system finishes installing the backup image, the installation program reconfigures the ODM on the target system. If the target system does not have exactly the same hardware configuration as the source system, the program might modify device attributes in the following target system files:

- All files in `/etc/objrepos` beginning with `Cu`
- All files in the `/dev` directory.

Related information

Installing system backups

Implementing scheduled backups

This procedure describes how to develop and use a script to perform a weekly full backup and daily incremental backups of user files.

- The amount of data scheduled for backup cannot exceed one tape when using this script.
- Make sure the tape is loaded in the backup device before the **cron** command runs the script.
- Make sure the device is connected and available, especially when using scripts that run at night. Use the **lsdev -C | pg** command to check availability.
- Make sure the backup device has been cleaned recently to prevent errors.
- If you are backing up file systems that might be in use, unmount them first to prevent file system corruption.
- Check the file system before making the backup. Use the procedure described in File system verification or run the **fsck** command.

The script included in this procedure is intended only as a model and needs to be carefully tailored to the needs of the specific site.

Related concepts

Backup strategy

There are two methods of backing up large amounts of data.

Backing up file systems using the cron command

This procedure describes how to write a **crontab** script that you can pass to the **cron** command for execution.

The script backs up two user file systems, `/home/plan` and `/home/run`, on Monday through Saturday nights. Both file systems are backed up on one tape, and each morning a new tape is inserted for the next night. The Monday night backups are full archives (level 0). The backups on Tuesday through Saturday are incremental backups.

1. The first step in making the **crontab** script is to issue the **crontab** -e command. This opens an empty file where you can make the entries that are submitted to the **cron** script for execution each night (the default editor is **vi**). Type:

```
crontab -e
```

2. The following example shows the six **crontab** fields. Field 1 is for the minute, field 2 is for the hour on a 24-hour clock, field 3 is for the day of the month, and field 4 is for the month of the year. Fields 3 and 4 contain an * (asterisk) to show that the script runs every month on the day specified in the **day/wk** field. Field 5 is for the day of the week, and can also be specified with a range of days, for example, 1-6. Field 6 is for the shell command being run.

```
min hr day/mo mo/yr day/wk       shell command

0   2    *     *       1         backup -0 -uf /dev/rmt0.1 /home/plan
```

   The command line shown assumes that personnel at the site are available to respond to prompts when appropriate. The -0 (zero) flag for the **backup** command stands for level zero, or full backup. The **-u** flag updates the backup record in the /etc/dumpdates file and the **f** flag specifies the device name, a raw magnetic tape device 0.1 as in the example above.

3. Type a line similar to that in step 2 for each file system backed up on a specific day. The following example shows a full script that performs six days of backups on two file systems:

```
0 2 * * 1 backup -0 -uf/dev/rmt0.1 /home/plan
0 3 * * 1 backup -0 -uf/dev/rmt0.1 /home/run
0 2 * * 2 backup -1 -uf/dev/rmt0.1 /home/plan
0 3 * * 2 backup -1 -uf/dev/rmt0.1 /home/run
0 2 * * 3 backup -2 -uf/dev/rmt0.1 /home/plan
0 3 * * 3 backup -2 -uf/dev/rmt0.1 /home/run
0 2 * * 4 backup -3 -uf/dev/rmt0.1 /home/plan
0 3 * * 4 backup -3 -uf/dev/rmt0.1 /home/run
0 2 * * 5 backup -4 -uf/dev/rmt0.1 /home/plan
0 3 * * 5 backup -4 -uf/dev/rmt0.1 /home/run
0 2 * * 6 backup -5 -uf/dev/rmt0.1 /home/plan
0 3 * * 6 backup -5 -uf/dev/rmt0.1 /home/run
```

4. Save the file you created and exit the editor. The operating system passes the crontab file to the **cron** script.

Related information

rmt Special File

Backup of files on a DMAPI-managed JFS2 file system

There are options in the **tar** and **backbyinode** commands that allow you to back up the extended attributes (EAs).

With the **backbyinode** command on a DMAPI file system, only the data resident in the file system at the time the command is issued is backed up. The **backbyinode** command examines the current state of metadata to do its work. This can be advantageous with DMAPI, because it backs up the state of the managed file system. However, any offline data will not be backed up.

To back up all of the data in a DMAPI file system, use a command that reads entire files, such as the **tar** command. This can cause a DMAPI-enabled application to restore data for every file accessed by the **tar** command, moving data back and forth between secondary and tertiary storage, so there can be performance implications.

Formatting diskettes (format or fdformat command)

You can format diskettes in the diskette drive specified by the *Device* parameter (the /dev/rfd0 device by default) with the **format** and **fdformat** commands.

Attention: Formatting a diskette destroys any existing data on that diskette.

The **format** command determines the device type, which is one of the following:

- 5.25-inch low-density diskette (360 KB) containing 40x2 tracks, each with 9 sectors

- 5.25-inch high-capacity diskette (1.2 MB) containing 80x2 tracks, each with 15 sectors
- 3.5-inch low-density diskette (720 KB) containing 80x2 tracks, each with 9 sectors
- 3.5-inch high-capacity diskette (2.88 MB) containing 80x2 tracks, each with 36 sectors

The sector size is 512 bytes for all diskette types.

Use the **format** command to format a diskette for high density unless the *Device* parameter specifies a different density.

Use the **fdformat** command to format a diskette for low density unless the **-h** flag is specified. The *Device* parameter specifies the device containing the diskette to be formatted (such as the `/dev/rfd0` device for drive 0).

Before formatting a diskette, the **format** and **fdformat** commands prompt for verification. This allows you to end the operation cleanly if necessary.

See the following examples:

- To format a diskette in the `/dev/rfd0` device, type the following:

```
format -d /dev/rfd0
```

- To format a diskette without checking for bad tracks, type the following:

```
format -f
```

- To format a 360 KB diskette in a 5.25-inch, 1.2 MB diskette drive in the `/dev/rfd1` device, type the following:

```
format -l -d /dev/rfd1
```

- To force high-density formatting of a diskette when using the **fdformat** command, type the following:

```
fdformat -h
```

Checking the integrity of a file system (fsck command)

Use the **fsck** command to check and interactively repair inconsistent file systems.

It is important to run this command on every file system as part of system initialization. You must be able to read the device file on which the file system resides (for example, the `/dev/hd0` device). Normally, the file system is consistent, and the **fsck** command merely reports on the number of files, used blocks, and free blocks in the file system. If the file system is inconsistent, the **fsck** command displays information about the inconsistencies found and prompts you for permission to repair them. The **fsck** command is conservative in its repair efforts and tries to avoid actions that might result in the loss of valid data. In certain cases, however, the **fsck** command recommends the destruction of a damaged file.

Attention: Always run the **fsck** command on file systems after a system malfunction. Corrective actions can result in some loss of data. The default action for each consistency correction is to wait for the operator to type `yes` or `no`. If you do not have *write* permission for an affected file, the **fsck** command will default to a no response.

See the following examples:

- To check all the default file systems, type the following:

```
fsck
```

  This form of the **fsck** command asks you for permission before making any changes to a file system.
- To fix minor problems automatically with the default file systems, type the following:

```
fsck -p
```

- To check the /dev/hd1 file system , type the following:

```
fsck /dev/hd1
```

  This checks the unmounted file system located on the /dev/hd1 device.

Note: The **fsck** command does not make corrections to a mounted file system.

Copying to or from diskettes (flcopy command)

Use the **flcopy** command to copy a diskette (opened as /dev/rfd0) to a file named floppy created in the current directory.

The message Change floppy, hit return when done displays as needed. The **flcopy** command then copies the floppy file to the diskette.

See the following examples:

- To copy /dev/rfd1 to the floppy file in the current directory, type the following:

```
flcopy -f /dev/rfd1 -r
```

- To copy the first 100 tracks of the diskette, type the following:

```
flcopy -f /dev/rfd1 -t 100
```

Copying files to tape or disk (cpio -o command)

Use the **cpio -o** command to read file path names from standard input and copy these files to standard output, along with path names and status information.

Path names cannot exceed 128 characters. Avoid giving the **cpio** command path names made up of many uniquely linked files because it might not have enough memory to keep track of the path names and would lose linking information.

See the following examples:

- To copy files in the current directory whose names end with .c onto diskette, type the following:

```
ls *.c | cpio -ov >/dev/rfd0
```

  The **-v** flag displays the names of each file.

- To copy the current directory and all subdirectories onto diskette, type the following:

```
find . -print | cpio -ov >/dev/rfd0
```

  This saves the directory tree that starts with the current directory (.) and includes all of its subdirectories and files.

- To use a shorter command string, type the following:

```
find . -cpio /dev/rfd0 -print
```

  The -print entry displays the name of each file as it is copied.

Copying files from tape or disk (cpio -i command)

Use the **cpio -i** command to read from standard input an archive file created by the **cpio -o** command and copy from it the files with names that match the *Pattern* parameter.

These files are copied into the current directory tree. You can list more than one *Pattern* parameter by using the file name notation described in the **ksh** command. The default for the *Pattern* parameter is an asterisk (*), which selects all files in the current directory. In an expression such as [a-z], the hyphen (-) means *through*, according to the current collating sequence.

Note: The patterns "*.c" and "*.o" must be enclosed in quotation marks to prevent the shell from treating the asterisk (*) as a pattern-matching character. This is a special case in which the **cpio** command itself decodes the pattern-matching characters.

See the following examples:

- To list the files that have been saved onto a diskette with the **cpio** command, type the following:

```
cpio -itv </dev/rfd0
```

  This displays the table of contents of the data previously saved onto the /dev/rfd0 file in the **cpio** command format. The listing is similar to the long directory listing produced by the **ls -l** command.
- To list only the file path names, use only the **-it** flags.
- To copy the files previously saved with the **cpio** command from a diskette, type the following:

```
cpio -idmv </dev/rfd0
```

  This copies the files previously saved onto the /dev/rfd0 file by the **cpio** command back into the file system (specify the **-i** flag). The **-d** flag allows the **cpio** command to create the appropriate directories if a directory tree is saved. The **-m** flag maintains the last modification time in effect when the files are saved. The **-v** flag causes the **cpio** command to display the name of each file as it is copied.
- To copy selected files from diskette, type the following:

```
cpio -i "*.c" "*.o" </dev/rfd0
```

  This copies the files that end with .c or .o from diskette.

Copying to or from tapes (tcopy command)

Use the **tcopy** command to copy magnetic tapes.

For example, to copy from one streaming tape to a 9-track tape, type the following:

```
tcopy /dev/rmt0 /dev/rmt8
```

Checking the integrity of a tape (tapechk command)

Use the **tapechk** command to perform rudimentary consistency checking on an attached streaming tape device.

Some hardware malfunctions of a streaming tape drive can be detected by simply reading a tape. The **tapechk** command provides a way to perform tape reads at the file level.

For example, to check the first three files on a streaming tape device, type the following:

```
tapechk 3
```

Archiving files (tar command)

The archive backup method is used for a copy of one or more files, or an entire database that is saved for future reference, historical purposes, or for recovery if the original data is damaged or lost.

Usually, an archive is used when that specific data is removed from the system.

Use the **tar** command to write files to or retrieve files from an archive storage. The **tar** command looks for archives on the default device (usually tape), unless you specify another device.

When writing to an archive, the **tar** command uses a temporary file (the /tmp/tar* file) and maintains in memory a table of files with several links. You receive an error message if the **tar** command cannot create the temporary file or if there is not enough memory available to hold the link tables.

See the following examples:

- To write the file1 and file2 files to a new archive on the default tape drive, type the following:

```
tar -c file1 file2
```

- To extract all files in the /tmp directory from the archive file on the /dev/rmt2 tape device and use the time of extraction as the modification time, type the following:

```
tar -xm -f/dev/rmt2 /tmp
```

- To display the names of the files in the out.tar disk archive file from the current directory, type the following:

```
tar -vtf out.tar
```

File backup

Use either the **backup** command or the **smit** command to create copies of your files on backup media, such as a magnetic tape or diskette.

Attention: If you attempt to back up a mounted file system, a message displays. The **backup** command continues, but inconsistencies in the file system can occur. This situation does not apply to the root (/) file system.

The copies you created with the **backup** command or the **smit** command are in one of the following backup formats:

- Specific files backed up by name, using the **-i** flag.
- Entire file system backed up by i-node number, using the *-Level* and *FileSystem* parameters.

  Note:

  – The possibility of data corruption always exists when a file is modified during system backup. Therefore, make sure that system activity is at a minimum during the system backup procedure.
  – If a backup is made to 8-mm tape with the device block size set to 0 (zero), it is not possible to directly restore data from the tape. If you have done backups with the 0 setting, you can restore data from them by using special procedures described under the **restore** command.

Attention: Be sure the flags you specify match the backup media.

Backing up files using the backup command

Use the **backup** command to create copies of your files on backup media.

For example, to back up selected files in your **$HOME** directory by name, type the following:

```
find $HOME -print | backup -i -v
```

The **-i** flag prompts the system to read from standard input the names of files to be backed up. The **find** command generates a list of files in the user's directory. This list is piped to the **backup** command as standard input. The **-v** flag displays a progress report as each file is copied. The files are backed up on the default backup device for the local system.

See the following examples:

- To back up the root file system, type the following:

```
backup -0 -u /
```

  The 0 level and the / tell the system to back up the / (root) file system. The file system is backed up to the /dev/rfd0 file. The **-u** flag tells the system to update the current backup level record in the /etc/dumpdates file.

- To back up all files in the / (root) file system that were modified since the last 0 level backup, type the following:

```
backup -1 -u /
```

Backing up files using the smit command

Use the **smit** command to run the **backup** command, which creates copies of your files on backup media.

1. At the prompt, type the following:

```
smit backup
```

2. Type the path name of the directory on which the file system is normally mounted in the **DIRECTORY full pathname** field:

```
/home/bill
```

3. In the **BACKUP** device or **FILE** fields, enter the output device name, as in the following example for a raw magnetic tape device:

```
/dev/rmt0
```

4. Use the Tab key to toggle the optional **REPORT each phase of the backup** field if you want error messages printed to the screen.
5. In a system management environment, use the default for the **MAX number of blocks to write on backup medium** field because this field does not apply to tape backups.
6. Press Enter to back up the named directory or file system.
7. Run the **restore -t** command.

   If this command generates an error message, you must repeat the entire backup.

Shutting down the system

The **shutdown** command is the safest and most thorough way to halt the operating system.

You might want to shut down your system:

- After installing new software or changing the configuration for existing software
- When a hardware problem exists
- When the system is irrevocably hung
- When system performance is degraded
- When the file system is possibly corrupt.

When you designate the appropriate flags, this command notifies users that the system is about to go down, kills all existing processes, unmounts file systems, and halts the system. See **shutdown** for more information.

Review the following information for details on specific shutdown situations:

Shutting down the system without rebooting

There are two ways of shutting down the system with no reboot.

You can use two methods to shut down the system without rebooting: the SMIT fastpath, or the **shutdown** command.

Prerequisites

You must have root user authority to shut down the system.

To shut down the system using SMIT:

1. Log in as root.
2. At the command prompt, type:

```
smit shutdown
```

To shut down the system using the **shutdown** command:

1. Log in as root.
2. At the command prompt, type:

```
shutdown
```

Shutting down the system to single-user mode

In some cases, you might need to shut down the system and enter single-user mode to perform software maintenance and diagnostics.

1. Type `cd /` to change to the root directory.

   You must be in the root directory to shut down the system to single-user mode to ensure that file systems are unmounted cleanly.
2. Type `shutdown -m`.

   The system shuts down to single-user mode.

A system prompt displays and you can perform maintenance activities.

Shutting down the system in an emergency

Use the **shutdown** command to stop the system quickly without notifying other users.

You can use the **shutdown** command to shut down the system under emergency conditions.

Type `shutdown -F`. The **-F** flag instructs the **shutdown** command to bypass sending messages to other users and shut down the system as quickly as possible.

System environment

The system environment is primarily the set of variables that define or control certain aspects of process execution.

They are set or reset each time a shell is started. From the system-management point of view, it is important to ensure the user is set up with the correct values at log in. Most of these variables are set during system initialization. Their definitions are read from the `/etc/profile` file or set by default.

Profiles

The shell uses two types of profile files when you log in to the operating system.

The shell evaluates the commands contained in the files and then runs the commands to set up your system environment. The files have similar functions except that the `/etc/profile` file controls profile variables for all users on a system whereas the `.profile` file allows you to customize your own environment.

The following profile and system environment information is provided:

- `/etc/profile` file
- `.profile` file
- System environment variable setup
- Changing the Message of the Day
- “Time data manipulation services” on page 48.

/etc/profile file

The first file that the operating system uses at login time is the `/etc/profile` file. This file controls system-wide default variables such as:

- Export variables
- File creation mask (umask)
- Terminal types

- Mail messages to indicate when new mail has arrived.

The system administrator configures the `profile` file for all users on the system. Only the system administrator can change this file.

.profile File

The second file that the operating system uses at login time is the `.profile` file. The `.profile` file is present in your home ($HOME) directory and enables you to customize your individual working environment. The `.profile` file also overrides commands and variables set in the `/etc/profile` file. Because the `.profile` file is hidden, use the **ls -a** command to list it. Use the `.profile` file to control the following defaults:

- Shells to open
- Prompt appearance
- Environment variables (for example, search path variables)
- Keyboard sound

The following example shows a typical `.profile` file:

```
PATH=/usr/bin:/etc:/home/bin1:/usr/lpp/tps4.0/user:/home/gsc/bin::
epath=/home/gsc/e3:
export PATH epath
csh
```

This example has defined two paths (PATH and epath), exported them, and opened a C shell (csh).

You can also use the `.profile` file (or if it is not present, the `.profile` file) to determine login shell variables. You can also customize other shell environments. For example, use the `.chsrc` and `.kshrc` files to tailor a C shell and a Korn shell, respectively, when each type shell is started.

Time data manipulation services

The time functions access and reformat the current system date and time.

You do not need to specify any special flag to the compiler to use the time functions. Include the header file for these functions in the program. To include a header file, use the following statement:

```
#include <time.h>
```

The time services are the following:

Item	Description
adjtime	Corrects the time to allow synchronization of the system clock.
ctime, localtime, gmtime, mktime, difftime, asctime, tzset	Converts date and time to string representation.
getinterval, incinterval, absinterval, resinc, resabs, alarm, ualarm, getitimer, setitimer	Manipulates the expiration time of interval timers.
gettimer, settimer, restimer, stime, time	Gets or sets the current value for the specified systemwide timer.
gettimerid	Allocates a per-process interval timer.
gettimeofday, settimeofday, ftime	Gets and sets date and time.
nsleep, usleep, sleep	Suspends a current process from running.
reltimerid	Releases a previously allocated interval timer.

Filesets and hardware needed for 64-bit mode

The kernel runs in 64-bit mode, allowing fast access to large amounts of data and efficient handling of 64-bit data types.

The base operating system 64-bit runtime fileset is `bos.64bit`. Installing `bos.64bit` also installs the `/etc/methods/cfg64` file. The `/etc/methods/cfg64` file is a command that enables the 64-bit runtime environment. This command is invoked by the `rc.boot` script during phase 3 of the boot process.

Beginning with AIX 6.1, the 32-bit kernel has been deprecated. Installing the AIX 6.1 base operating system enables the 64-bit mode.

Note: Hardware must be 64-bit capable to run AIX 6.1. The following RS/6000 models use 604e processors and are not 64-bit capable:

- 7025 F50 Series
- 7026 H50 Series
- 9076 H50 Series
- 7043 150 Series
- 7046 B50 Series

To verify the capability of your processor, run the following command:

```
/usr/sbin/prtconf -c
```

The **prtconf** command returns either 32 or 64, depending on the capability of your processor. If your system does not have the **prtconf** command, you can use the **bootinfo** command with the `-y` flag.

Hardware required for 64-bit mode

You must have 64-bit hardware to run 64-bit applications.

To determine whether your system has 32-bit or 64-bit hardware architecture:

1. Log in as a root user.
2. At the command line, enter `bootinfo -y`.

This produces the output of either **32** or **64**, depending on whether the hardware architecture is 32-bit or 64-bit. In addition, if you enter `lsattr -El proc0` at any version of AIX, the type of processor for your server displays.

32-bit and 64-bit performance comparisons

In most cases, running 32-bit applications on 64-bit hardware is not a problem, because 64-bit hardware can run both 64-bit and 32-bit software. However, 32-bit hardware cannot run 64-bit software.

To find out if any performance issues exist for applications that are running on the system, refer to those application's user guides for their recommended running environment.

Dynamic Processor Deallocation

AIX can detect and automatically stop using a faulty processor.

Starting with machine type 7044 model 270, the hardware of all systems with two or more processors is able to detect correctable errors, which are gathered by the firmware. These errors are not fatal and, as long as they remain rare occurrences, can be safely ignored. However, when a pattern of failures seems to be developing on a specific processor, this pattern might indicate that this component is likely to exhibit a fatal failure in the near future. This prediction is made by the firmware based on the failure rates and threshold analysis.

On these systems, AIX implements continuous hardware surveillance and regularly polls the firmware for hardware errors. When the number of processor errors hits a threshold and the firmware recognizes that

there is a distinct probability that this system component will fail, the firmware returns an error report. In all cases, the error is logged in the system error log. In addition, on multiprocessor systems, depending on the type of failure, AIX attempts to stop using the untrustworthy processor and deallocate it. This feature is called *Dynamic Processor Deallocation*.

At this point, the processor is also flagged by the firmware for persistent deallocation for subsequent reboots, until maintenance personnel replaces the processor.

Processor deallocation impacts to applications

Processor deallocation is transparent for the vast majority of applications, including drivers and kernel extensions. However, you can use the published interfaces to determine whether an application or kernel extension is running on a multiprocessor machine, find out how many processors there are, and bind threads to specific processors.

The bindprocessor interface for binding processes or threads to processors uses bind CPU numbers. The bind CPU numbers are in the range [0..*N*-1] where *N* is the total number of CPUs. To avoid breaking applications or kernel extensions that assume no "holes" in the CPU numbering, AIX always makes it appear for applications as if it is the "last" (highest numbered) bind CPU to be deallocated. For instance, on an 8-way SMP, the bind CPU numbers are [0..7]. If one processor is deallocated, the total number of available CPUs becomes 7, and they are numbered [0..6]. Externally, it looks like CPU 7 has disappeared, regardless of which physical processor failed.

Note: In the rest of this description, the term *CPU* is used for the logical entity and the term *processor* for the physical entity.

Potentially, applications or kernel extensions that are binding processes or threads could be broken if AIX silently terminated their bound threads or forcefully moved them to another CPU when one of the processors needs to be deallocated. Dynamic Processor Deallocation provides programming interfaces so that such applications and kernel extensions can be notified that a processor deallocation is about to happen. When these applications and kernel extensions receive notification, they are responsible for moving their bound threads and associated resources (such as timer request blocks) away from the last bind CPU ID and for adapting themselves to the new CPU configuration.

After notification, if some threads remain bound to the last bind CPU ID, the deallocation is aborted, the aborted deallocation is logged in the error log, and AIX continues using the ailing processor. When the processor ultimately fails, it causes a total system failure. Therefore, it is important that applications or kernel extensions receive notification of an impending processor deallocation and act on this notice.

Even in the rare cases that the deallocation cannot go through, Dynamic Processor Deallocation still gives advanced warning to system administrators. By recording the error in the error log, it gives them a chance to schedule a maintenance operation on the system to replace the ailing component before a global system failure occurs.

Processor deallocation process

AIX can stop a failing processor by deallocating it.

The typical flow of events for processor deallocation is as follows:

1. The firmware detects that a recoverable error threshold has been reached by one of the processors.
2. The firmware error report is logged in the system error log, and, when AIX is executing on a machine that supports processor deallocation, AIX starts the deallocation process.
3. AIX notifies non-kernel processes and threads bound to the last bind CPU.
4. AIX waits up to ten minutes for all the bound threads to move away from the last bind CPU. If threads remain bound, AIX aborts the deallocation.
5. If all processes or threads are unbound from the ailing processor, the previously registered High Availability Event Handlers (HAEHs) are invoked. An HAEH might return an error that aborts the deallocation.
6. Unless aborted, the deallocation process ultimately stops the failing processor.

If there is a failure at any point of the deallocation, the failure and its cause are logged. The system administrator can look at the error log, take corrective action (when possible) and restart the deallocation. For instance, if the deallocation was aborted because an application did not unbind its bound threads, the system administrator can stop the application, restart the deallocation, and then restart the application.

Enabling Dynamic Processor Deallocation

If your machine supports Dynamic Processor Deallocation, you can use SMIT or system commands to turn the feature **on** or **off**.

Dynamic Processor Deallocation is enabled by default during installation, provided the machine has the correct hardware and firmware to support it.

SMIT fastpath procedure

1. With root authority, type `smit system` at the system prompt, then press Enter.
2. In the **Systems Environment** window, select **Change / Show Characteristics of Operating System**.
3. Use the SMIT dialogs to complete the task.

To obtain additional information for completing the task, you can select the F1 Help key in the SMIT dialogs.

Commands procedure

With root authority, you can use the following commands to work with the Dynamic Processor Deallocation:

- Use the **chdev** command to change the characteristics of the device specified.
- If the processor deallocation fails for any reason, you can use the **ha_star** command to restart it after it has been fixed.
- Use the **errpt** command to generate a report of logged errors.

Methods of turning processor deallocation on and off

Dynamic Processor Deallocation can be enabled or disabled by changing the value of the `cpuguard` attribute of the ODM object `sys0`.

The possible values for the attribute are `enable` and `disable`.

The default is `enabled` (the attribute `cpuguard` has a value of `enable`). System administrators who want to disable this feature must use either the system menus, the SMIT **System Environments** menu, or the **chdev** command. (In previous AIX versions, the default was `disabled`.)

Note: If processor deallocation is turned off (disabled), the errors are still logged. The error log will contain an error such as `CPU_FAILURE_PREDICTED`, indicating that AIX was notified of a problem with a CPU.

Restarting an aborted processor deallocation

Sometimes the processor deallocation fails because an application did not move its bound threads away from the last logical CPU.

Once this problem has been fixed, either by unbinding (when it is safe to do so) or by stopping the application, the system administrator can restart the processor deallocation process using the **ha_star** command.

The syntax for this command is:

```
ha_star -C
```

where **-C** is for a CPU predictive failure event.

Processor state considerations

There are several things you should consider about processor states.

Physical processors are represented in the ODM database by objects named **proc***n* where *n* is a decimal number that represents the physical processor number. Like any other device represented in the ODM database, processor objects have a state, such as Defined/Available, and attributes.

The state of a **proc** object is always Available as long as the corresponding processor is present, regardless of whether it is usable. The **state** attribute of a **proc** object indicates if the processor is used and, if not, the reason. This attribute can have three values:

Item	Description
enable	The processor is used.
disable	The processor has been dynamically deallocated.
faulty	The processor was declared defective by the firmware at startup time.

If an ailing processor is successfully deallocated, its state goes from **enable** to **disable**. Independently of AIX, this processor is also flagged in the firmware as defective. Upon reboot, the deallocated processor will not be available and will have its state set to **faulty**. The ODM **proc** object, however, is still marked Available. You must physically remove the defective CPU from the system board or remove the CPU board (if possible) for the **proc** object to change to Defined.

In the following example, processor **proc4** is working correctly and is being used by the operating system, as shown in the following output:

```
# lsattr -EH -l proc4
attribute    value            description        user_settable

state        enable            Processor state        False
type        PowerPC_RS64-III    Processor type        False
    #
```

When processor **proc4** gets a predictive failure, it gets deallocated by the operating system, as shown in the following:

```
# lsattr -EH -l proc4
attribute    value            description        user_settable

state        disable            Processor state        False
type        PowerPC_RS64-III    Processor type        False
    #
```

At the next system restart, processor **proc4** is reported by firmware as defective, as shown in the following:

```
# lsattr -EH -l proc4
attribute    value            description        user_settable

state        faulty            Processor state        False
type        PowerPC_RS64-III    Processor type        False
    #
```

But the status of processor **proc4** remains Available, as shown in the following:

```
# lsdev -CH -l proc4
name        status            location        description

proc4        Available        00-04            Processor
#
```

Deallocation error log entries

Three different error log messages are associated with CPU deallocation.

The following are examples.

errpt short format - summary

The following is an example of entries displayed by the **errpt** command (without options):

```
# errpt
IDENTIFIER      TIMESTAMP       T    C    RESOURCE_NAME    DESCRIPTION
804E987A        1008161399      I    O    proc4            CPU DEALLOCATED
8470267F        1008161299      T    S    proc4            CPU DEALLOCATION ABORTED
1B963892        1008160299      P    H    proc4            CPU FAILURE PREDICTED
#
```

- If processor deallocation is enabled, a `CPU FAILURE PREDICTED` message is always followed by either a `CPU DEALLOCATED` message or a `CPU DEALLOCATION ABORTED` message.
- If processor deallocation is not enabled, only the `CPU FAILURE PREDICTED` message is logged. Enabling processor deallocation any time after one or more `CPU FAILURE PREDICTED` messages have been logged initiates the deallocation process and results in a success or failure error log entry, as described above, for each processor reported failing.

errpt long format - detailed description

The following is the form of output obtained with **errpt -a**:

- `CPU_FAIL_PREDICTED`

  Error description: Predictive Processor Failure

  This error indicates that the hardware detected that a processor has a high probability to fail in a near future. It is always logged whether or not processor deallocation is enabled.

  DETAIL DATA: *Physical processor number*, *location*

  Example error log entry - long form

```
    LABEL:            CPU_FAIL_PREDICTED
    IDENTIFIER:        1655419A

    Date/Time:        Thu Sep 30 13:42:11
    Sequence Number:    53
    Machine Id:        00002F0E4C00
    Node Id:        auntbea
    Class:            H
    Type:            PEND
    Resource Name:         proc25
    Resource Class:         processor
    Resource Type:         proc_rspc
    Location:        00-25

    Description
    CPU FAILURE PREDICTED

    Probable Causes
    CPU FAILURE

    Failure Causes
    CPU FAILURE

        Recommended Actions
        ENSURE CPU GARD MODE IS ENABLED
        RUN SYSTEM DIAGNOSTICS.

    Detail Data
    PROBLEM DATA
    0144    1000    0000    003A    8E00    9100    1842    1100    1999    0930    4019
    0000    0000    0000    0000    0000
    0000    0000    0000    0000    0000    0000    0000    0000    4942    4D00    5531
    2E31    2D50    312D    4332    0000
    0002    0000    0000    0000    0000    0000    0000    0000    0000    0000    0000
    0000    0000    0000    0000    0000
    0000    0000    0000    0000    0000    0000    0000    0000    0000    0000    0000
    0000    0000    0000    0000    0000
    ...    ...    ...    ...    ...
```

- `CPU_DEALLOC_SUCCESS`

Error Description: A processor has been successfully deallocated after detection of a predictive processor failure. This message is logged when processor deallocation is enabled, and when the CPU has been successfully deallocated.

DETAIL DATA: *Logical CPU number of deallocated processor*.

Example: error log entry - long form:

```
LABEL:            CPU_DEALLOC_SUCCESS
IDENTIFIER:        804E987A

Date/Time:        Thu Sep 30 13:44:13
Sequence Number:    63
Machine Id:        00002F0E4C00
Node Id:        auntbea
Class:            O
Type:            INFO
Resource Name:        proc24

Description
CPU DEALLOCATED


    Recommended Actions
    MAINTENANCE IS REQUIRED BECAUSE OF CPU FAILURE

Detail Data
LOGICAL DEALLOCATED CPU NUMBER

    0
```

In this example, **proc24** was successfully deallocated and was logical CPU **0** when the failure occurred.

- CPU_DEALLOC_FAIL

Error Description: A processor deallocation, due to a predictive processor failure, was not successful. This message is logged when CPU deallocation is enabled, and when the CPU has not been successfully deallocated.

DETAIL DATA: *Reason code, logical CPU number, additional information depending of the type of failure.*

The reason code is a numeric hexadecimal value. The possible reason codes are:

Item	Description
2	One or more processes/threads remain bound to the last logical CPU. In this case, the detailed data give the PIDs of the offending processes.
3	A registered driver or kernel extension returned an error when notified. In this case, the detailed data field contains the name of the offending driver or kernel extension (ASCII encoded).
4	Deallocating a processor causes the machine to have less than two available CPUs. This operating system does not deallocate more than *N*-2 processors on an *N*-way machine to avoid confusing applications or kernel extensions using the total number of available processors to determine whether they are running on a Uni Processor (UP) system where it is safe to skip the use of multiprocessor locks, or a Symmetric Multi Processor (SMP).
200 (0xC8)	Processor deallocation is disabled (the ODM attribute **cpuguard** has a value of disable). You normally do not see this error unless you start **ha_star** manually.

Examples: error log entries - long format

Example 1:

```
LABEL:            CPU_DEALLOC_ABORTED
IDENTIFIER:        8470267F
Date/Time:        Thu Sep 30 13:41:10
```

```
    Sequence Number:    50
    Machine Id:        00002F0E4C00
    Node Id:        auntbea
    Class:            S
    Type:            TEMP
    Resource Name:        proc26

Description
CPU DEALLOCATION ABORTED

Probable Causes
SOFTWARE PROGRAM

Failure Causes
SOFTWARE PROGRAM

    Recommended Actions
    MAINTENANCE IS REQUIRED BECAUSE OF CPU FAILURE
    SEE USER DOCUMENTATION FOR CPU GARD

Detail Data
DEALLOCATION ABORTED CAUSE
0000 0003
DEALLOCATION ABORTED DATA
6676 6861 6568 3200
```

In this example, the deallocation for **proc26** failed. The reason code 3 means that a kernel extension returned an error to the kernel notification routine. The `DEALLOCATION ABORTED DATA` above spells **fvhaeh2**, which is the name the extension used when registering with the kernel.

Example 2:

```
    LABEL:            CPU_DEALLOC_ABORTED
    IDENTIFIER:        8470267F
    Date/Time:        Thu Sep 30 14:00:22
    Sequence Number:    71
    Machine Id:        00002F0E4C00
    Node Id:        auntbea
    Class:            S
    Type:            TEMP
    Resource Name:        proc19

Description
CPU DEALLOCATION ABORTED

Probable Causes
SOFTWARE PROGRAM

Failure Causes
SOFTWARE PROGRAM

    Recommended Actions
    MAINTENANCE IS REQUIRED BECAUSE OF CPU FAILURE;
    SEE USER DOCUMENTATION FOR CPU GARD

Detail Data
DEALLOCATION ABORTED CAUSE
0000 0002
DEALLOCATION ABORTED DATA
0000 0000 0000 4F4A
```

In this example, the deallocation for **proc19** failed. The reason code 2 indicates thread(s) were bound to the last logical processor and did not unbind after receiving the `SIGCPUFAIL` signal. The `DEALLOCATION ABORTED DATA` shows that these threads belonged to process **0x4F4A**.

Options of the **ps** command (`-o THREAD, -o BND`) allow you to list all threads or processes along with the number of the CPU they are bound to, when applicable.

Example 3:

```
    LABEL:            CPU_DEALLOC_ABORTED
    IDENTIFIER:        8470267F

    Date/Time:        Thu Sep 30 14:37:34
    Sequence Number:    106
    Machine Id:        00002F0E4C00
    Node Id:        auntbea
```

```
    Class:            S
    Type:            TEMP
    Resource Name:        proc2

Description
CPU DEALLOCATION ABORTED

Probable Causes
SOFTWARE PROGRAM

Failure Causes
SOFTWARE PROGRAM

    Recommended Actions
    MAINTENANCE IS REQUIRED BECAUSE OF CPU FAILURE
    SEE USER DOCUMENTATION FOR CPU GARD

Detail Data
DEALLOCATION ABORTED CAUSE
0000 0004
DEALLOCATION ABORTED DATA
0000 0000 0000 0000
```

In this example, the deallocation of **proc2** failed because there were two or fewer active processors at the time of failure (reason code 4).

System environment variable setup

The system environment is primarily the set of variables that define or control certain aspects of process execution.

They are set or reset each time a shell is started. From the system-management point of view, it is important to ensure the user is set up with the correct values at login. Most of these variables are set during system initialization. Their definitions are read from the `/etc/profile` file or set by default.

Testing the system battery

If your system is losing track of time, the cause might be a depleted or disconnected battery.

1. To determine the status of your system battery, type the following **diag** command:

   ```
   diag -B -c
   ```

2. When the Diagnostics main menu appears, select the **Problem Determination** option.

   If the battery is disconnected or depleted, a problem menu will be displayed with a service request number (SRN). Record the SRN on Item 4 of the Problem Summary Form and report the problem to your hardware service organization.

If your system battery is operational, your system time might have been reset incorrectly because either the **date** or **setclock** command was run incorrectly or unsuccessfully.

Related concepts

Setting up the system clock
The system clock records the time of system events, allows you to schedule system events (such as running hardware diagnostics at 3:00 a.m.), and tells when you first created or last saved files.

Setting up the system clock

The system clock records the time of system events, allows you to schedule system events (such as running hardware diagnostics at 3:00 a.m.), and tells when you first created or last saved files.

Use the **date** command to set your system clock. Use the **setclock** command to set the time and date by contacting a time server.

Related tasks

Testing the system battery

If your system is losing track of time, the cause might be a depleted or disconnected battery.

date command
The **date** command displays or sets the date and time.

Enter the following command to determine what your system recognizes as the current date and time:

```
/usr/bin/date
```

Attention: Do not change the date when the system is running with more than one user.

The following formats can be used when setting the date with the *Date* parameter:

- *mmddHHMM*[*YYyy*] (default)
- *mmddHHMM*[*yy*]

The variables to the *Date* parameter are defined as follows:

Item	Description
mm	Specifies the number of the month.
dd	Specifies the number of the day in the month.
HH	Specifies the hour in the day (using a 24-hour clock).
MM	Specifies the minute number.
YY	Specifies the first two digits of a four-digit year.
yy	Specifies the last two numbers of the year.

With root authority, you can use the **date** command to set the current date and time. For example:

```
date 021714252002
```

Sets the date to Feb. 17, 2002, and time to 14:25.

setclock command
The **setclock** command displays or sets the time and date by requesting the current time from a time server on a network.

To display your system's date and time, enter:

```
/usr/sbin/setclock
```

The **setclock** command takes the first response from the time server, converts the calendar clock reading found there, and shows the local date and time. If no time server responds, or if the network is not operational, the **setclock** command displays a message to that effect and leaves the date and time settings unchanged.

Note: Any host running the **inetd** daemon can act as a time server.

With root authority, you can use the **setclock** command to send an Internet TIME service request to a time server host and sets the local date and time accordingly. For example:

```
setclock TimeHost
```

Where *TimeHost* is the host name or IP address of the time server.

Related information
setclock command

Olson time zone support and setup

Beginning with AIX 6.1, support for time zone values consistent with the Olson database are provided.

The POSIX time zone specification supported in previous AIX releases, does not adequately handle changes to time zone rules such as daylight saving time. The Olson database maintains a historical record of time zone rules, so that if the rules change in a specific location, AIX interprets dates and time correctly both in the present and in the past.

Time zone definitions conforming to the POSIX specification are still supported and recognized by AIX. AIX checks the **TZ** environment variable to determine if the environment variable matches an Olson time zone value. If the **TZ** environment variable does not match an Olson time zone value, AIX then follows the POSIX specification rules.

For more details about the TZ environment variable, refer to Environment file.

To set the time zone using Olson defined values, use the following SMIT path: **System Environments** > **Change / Show Date, Time and Time Zone** > **Change Time Zone Using System Defined Values**.

Message of the day setup

The message of the day is displayed every time a user logs in to the system.

It is a convenient way to communicate information to all users, such as installed software version numbers or current system news. To change the message of the day, use your favorite editor to edit the `/etc/motd` file.

AIX usage metric data (SLM tags) for IBM License Metric Tool

The Software License Metric (SLM) tags generated by the AIX operating system serve as a usage metric data that is used by the IBM® License Metric Tool. The usage metric data records the `virtual CPU (vCPU)` information that represents the number of virtual CPUs that are online in the system.

To use the IBM License Metric Tool, you must install the `slm.rte` fileset that is available in the AIX operating system expansion pack and the fileset can also be downloaded from the website.

The IBM License Metric Tool generates a `vcpu.slmtag` Software License Metric Tag (SLMTAG) file that expands dynamically. The `vcpu.slmtag` file is located in the `/var/opt/slm` directory. This file is used by the BigFix Agent (BESClient) to incorporate the IBM License Metric Tool Agent software and to send the file to the IBM License Metric Tool server.

The IBM License Metric Tool can be used with the BigFix Agent (BESClient) that is installed and configured in the system. The BigFix Agent must be separately obtained from IBM License Metric Tool server and installed in the client. If the BigFix agent is not installed in the system, the SLMTAG file is created, but data is not sent to the IBM License Metric Tool server.

You can configure the IBM License Metric Tool with the environment variables that are defined in the `/etc/environment` file. At the time of installation, these variables are set to default values. The following configurable variables can be configured.

Variable	Details
SLM_VCPU_PERIOD_HOURS	Time period that is specified in hours, for adding a new metric entry. The default value is 720 hours (30 days). This value must be a multiple of 4. If you specify a value that is not a multiple of 4, the next multiple of 4 is considered by the IBM License Metric Tool. The minimum value is 4 hours.
SLM_VCPU_MAX_FSIZE	Maximum file size that is specified in bytes is the maximum size (in bytes) for the record file. If the file size exceeds this limit, the file is archived. The default value is 2097152 bytes (2 MB). The minimum value is 10 KB. If you specify a lower

Variable	Details
	value, the IBM License Metric Tool considers it as 10 KB.
SLM_VCPU_COUNT_ARCH	Number of archives retained by the tool. The default value is 4. The minimum value is 1. If you specify a lower value, the IBM License Metric Tool considers it as 1.

The usage metric data is displayed in the **Resource Utilization** page of the IBM License Metric Tool Server GUI. The usage metric data is displayed next to the host name as the VCPUmetric type and the COUNT subtype.

Related information

IBM License Metric Tool 9.2.0

AIX Runtime Expert

AIX Runtime Expert provides a simplified set of actions that can be used against a single consolidation for collecting, applying, and verifying the runtime environment for one or more AIX instances.

There are tools provided by AIX components, such as Reliability Availability Serviceability (RAS), Security, or Kernel, which allow you to change settings within each component layer in order to tune the operating system to a particular need or requirement. AIX Runtime Expert enables system-wide configuration by using an extendable framework to handle the many different configuration methods that currently exist in AIX.

AIX Runtime Expert executes multiple-component configuration commands as a single action using a configuration profile. You can use this profile to apply identical system settings across multiple systems. AIX Runtime Expert provides a simplified alternative for managing the runtime configuration of one or more systems, but it does not prevent the use of other methods to change system settings.

AIX Runtime Expert concepts

You must have basic knowledge of AIX Runtime Expert before you start using it.

AIX Runtime Expert base capabilities support configuration profile management and application for a single AIX system. To enable multiple system scalable consumption for a single profile, an LDAP based profile description can be discovered and consumed by AIX systems as they start or as the system is directed by administrative operations at the target AIX endpoints. Remote management for AIX Runtime Expert can only be done with the Network Install Manager (NIM) component. Using existing NIM functions, you can run AIX Runtime Expert remotely on several stand-alone NIM clients from a NIM master machine.

AIX Runtime Expert profiles

AIX Runtime Expert profiles are used to set values on a running system, extract values for a running system, and compare values against a running system or against another profile.

A profile describes one or more runtime configuration controls and their settings for the targeted functional area. A profile can represent a full set of controls or a subset of controls and their values. Configuration profiles are standard XML files. Using AIX Runtime Expert you can manage profiles and apply them on the defined system.

A profile can contain configuration parameters and tuning parameters without any values, like sample profiles. The purpose of a profile without any parameters is to extract the current systems values from the specified profile. Profiles containing at least one parameter without any values have the following limitations:

- Using the **artexset** command fails with an error.
- Using the **artexdiff** command returns a warning message for each parameter that has no value.

The value of a parameter in a profile can contain the following:

- No value
- A blob value, which is a base64 encoded binary data as an in-line text file. The blob value is used to replace existing files, like `/etc/motd` or `/etc/hosts`.
- A non-blob value, which is a value assigned to system configuration parameters, like an integer or string.

In the `/etc/security/artex/samples` directory you can view existing sample profiles. The sample profiles only contain parameter names that are supported by the default settings installed with AIX Runtime Expert. The parameters in the sample profiles do not have any values. Sample profiles are read only files. Use the sample profiles as a template to create new configuration profiles. You cannot apply existing samples to a running system.

The following examples are some of the base configuration commands that can be controlled through configuration profiles:

- Network configuration
  - no
  - mktcpip
- Kernel configuration
  - ioo
  - schedo
- RAS configuration
  - alog
- Security configuration
  - setsecattr

Example

The following example displays a configuration profile for different catalogs and sub-catalogs with assigned values for different parameters. You could edit this profile with any XML editor or use the **vi** command and change the existing values for the defined parameters.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Profile origin="get" version="1.0" date="2009-04-25T15:33:37Z">
<Catalog id="vmoParam">
<Parameter name="kernel_heap_psize" value="0" applyType="nextboot" reboot="true" />
<Parameter name="maxfree" value="1088" />
</Catalog>
<Catalog id="noParam">
<SubCat id="tcp_network">
<Parameter name="tcp_recvspace" value="16384" />
<Parameter name="tcp_sendspace" value="16384" />
</SubCat>
<SubCat id="general_network">
<Parameter name="use_sndbufpool" value="1" applyType="nextboot" reboot="true" />
</SubCat>
</Catalog>
<Catalog id="lvmoParam">
<Parameter name="max_vg_pbuf_count" value="0">
<Target class="vg" instance="rootvg" />
</Parameter>
<Parameter name="pv_pbuf_count" value="512">
<Target class="vg" instance="rootvg" />
</Parameter>
</Catalog>
```

Related tasks

Modifying AIX Runtime Expert profiles
AIX Runtime Expert profiles are XML files and can be modified with any XML editor or any text editor.

Creating AIX Runtime Expert profiles

Use existing samples in the `/etc/security/artex/samples` directory to create a new profile with the **artexget** command. The sample profiles are a template for you to create a profile that you can modify and save into a custom file.

Getting AIX Runtime Expert profile values
Use the **artexget** command to find information about a profile.

Applying AIX Runtime Expert profiles
To set a system with the configuration and tunable parameters from a profile, apply a profile using the **artexset** command.

AIX Runtime Expert catalogs

Catalogs are the mechanism that defines and specifies configuration controls that can be operated on by AIX Runtime Expert.

Catalogs are provided for the controls that are currently supported by the AIX Runtime Expert. Catalogs are definition files that map configuration profile values to parameters that run commands and configuration actions.

AIX Runtime Expert provides you with existing read-only catalogs, located in the `/etc/security/artex/catalogs` directory, that identify values that can be modified. Do not modify these catalogs.

Each catalog contains parameters for one component. However, some catalogs can contain parameters from more than one closely related components. The names of the catalogs describe the components that are contained in the catalog. The `<description>` XML element in each catalog provides a description of the catalog.

AIX Runtime Expert and LDAP

AIX Runtime Expert can retrieve profiles from the Lightweight Directory Access Protocol (LDAP) server.

The AIX Runtime Expert profiles must be stored as ibm-artexProfile objects and have the following mandatory attributes:

- Ibm-artexProfileName. The AIX Runtime Expert profile name.
- Ibm-artexProfileXMLData. The XML content of the AIX Runtime Expert profile that is stored as an octetString.

The AIX Runtime Expert schema must be installed on the LDAP server before storing any AIX Runtime Expert profiles. Setting up an LDAP server for AIX Runtime Expert is similar to setting up an LDAP server for user authentication. For more information about setting up LDAP, see Setting up an ITDS security information server.

Setting up an LDAP client for AIX Runtime Expert is similar to setting up an LDAP client for user authentication. For more information, view the Setting up an LDAP client topic. To set up an LDAP client, use the **mksecldap -c** command to correctly configure the **secldapclntd** daemon. AIX Runtime Expert relies on the **secldapclntd** daemon to access the LDAP server. By default, AIX Runtime Expert looks for profile entries under the identifier `DN: ou=artex,cn=AIXDATA`. You can customize this DN by updating the artexbasedn key in the `/etc/security/ldap/ldap.cfg` secldapclntd configuration file.

Uploading an AIX Runtime Expert profile

To upload an AIX Runtime Expert profile, you can either create an LDAP data interchange formatted (LDIF) file and use the **ldapadd** command or use an LDAP administration tool such as Tivoli® Directory Server Web Administration Tool.

The following is an example of a profile that is saved in LDIF:

```
dn: ou=artex,cn=AIXDATA
objectClass: organizationalUnit
objectClass: top
ou: artex

dn: ibm-artexProfileName=alogProfile.xml,ou=artex,cn=AIXDATA
objectClass: ibm-artexProfile
objectClass: top
```

```
ibm-artexProfileName: alogProfile.xml
ibm-artexProfileXMLData:< file:///etc/security/artex/samples/alogProfile.xml
```

The following is an example of uploading a profile using the **ldapadd** command and a sample LDIF file named sample.ldif:

```
ldapadd -c -h <ldaphost> -D cn=admin -w <password> -f sample.ldif
```

Related tasks

Creating AIX Runtime Expert profiles

Use existing samples in the `/etc/security/artex/samples` directory to create a new profile with the **artexget** command. The sample profiles are a template for you to create a profile that you can modify and save into a custom file.

Related information

IBM Security Directory Server

AIX Runtime Expert and RBAC

Role Based Access Control (RBAC) can be used to give non root users the ability to execute the AIX Runtime Expert commands.

AIX Runtime Expert authorizations

On installing the **artex.base.rte** fileset three system authorizations get created that allow different levels of access to the AIX Runtime Expert functionality:

- The **aix.system.config.artex.read** authorization allows the execution of the **artexlist** and **artexmerge** commands. The **artexget** and **artexdiff** commands are also allowed, but only to obtain the profile values. The values cannot be captured from the system (that is the **artexget** command cannot be run with the *–r*, *–n* or *–p* flags, and **artexdiff** command can only be run between two profiles).
- The **aix.system.config.artex.get** authorization allows all operations allowed by the **artex.system.config.read** authorization, and additionally allows the unrestricted execution of the **artexget** and **artexdiff** command.
- The **aix.system.config.artex.set** authorization allows all operations allowed by the **artex.system.config.get** authorization and additionally allows the execution of the **artexset** command.

AIX Runtime Expert roles

AIX Runtime Expert does not create any new role however the **artex.base.rte** filesets add the **aix.system.config.artex** authorization to the **SysConfig** role. Any user with **SysConfig** role or any enclosing role (such as the **isso** role) will be able to run the **artexlist**, **artexmerge**, **artexdiff**, **artexget** and **artexset** commands.

Restrictions

For security reasons, the use of the **ARTEX_CATALOG_PATH** environment variable is restricted to the root user. Non root users who are granted the right to execute the AIX Runtime Expert commands through the RBAC cannot use the **ARTEX_CATALOG_PATH** environment variable.

Administering AIX Runtime Expert

AIX Runtime Expert uses a few simple commands to create profiles, modify profiles, combine profiles, and apply profiles.

Configuring AIX Runtime Expert

AIX Runtime Expert uses the configuration file `/etc/security/artex/config/artex.conf`.

An entry in the configuration file consists of the name of a configuration option, followed by one or more spaces and a value. Blank lines and lines starting with a # sign are ignored.

The following options are supported:

Table 1. Configuration Options

Options	Description
ARTEX_CATALOG_PATH	Colon-separated list of directories searched for catalog files. This option is overridden by the **ARTEX_CATALOG_PATH** environment variable. Default path is `/etc/security/artex/catalogs`.
ARTEX_PROFILE_PATH	Colon-separated list of directories searched for profile files by the **artexlist** command if no directory is specified. This option is overridden by the **ARTEX_PROFILE_PATH** environment variable. Default path is `/etc/security/artex/samples`.
DEBUG_LOG_CATEGORY	Debug category for the log file. This option can be repeated to select multiple debug categories.
DEBUG_LOG_LEVEL	Debug level for the log file between 0 (no debug traces) and 3 (most verbose).
MAX_CMDS	Maximum number of external commands executed concurrently. External commands executed by AIX Runtime Expert are queued so that no more than MAX_CMDS external commands are executed simultaneously at any given time. Default is 10.

Creating AIX Runtime Expert profiles

Use existing samples in the `/etc/security/artex/samples` directory to create a new profile with the **artexget** command. The sample profiles are a template for you to create a profile that you can modify and save into a custom file.

To create a profile with all of the parameters supported by AIX Runtime Expert, complete the following steps:

1. Configure and tune your system to have the desired settings for a new profile.
2. Go to the samples directory: `/etc/security/artex/samples`
3. Run the following command to create a new profile named `custom_all.xml`:

   ```
   artexget -p all.xml > /directory_for_new_profile/custom_all.xml
   ```

   Note: The `custom_all.xml` profile can be used to configure other systems that have a similar current system configuration.

To create a profile for a specific component, such as network options, complete the following steps:

1. Configure and tune your system to have the desired settings for a new profile.
2. Go to the samples directory: `/etc/security/artex/samples`.

3. Create a new profile named `custom_no.xml` from the existing sample profile, `noProfile.xml`, by running the following command:

```
artexget -p noProfile.xml > /directory_for_new_profile/custom_no.xml
```

The newly created profiles can be customized by changing or removing the values of the parameters using an XML editor or any text editor.

The custom profiles can be uploaded to LDAP server to use from multiple AIX systems. To upload the profiles to LDAP server, use the tools provided by LDAP.

Related concepts

AIX Runtime Expert and LDAP
AIX Runtime Expert can retrieve profiles from the Lightweight Directory Access Protocol (LDAP) server.

AIX Runtime Expert profiles
AIX Runtime Expert profiles are used to set values on a running system, extract values for a running system, and compare values against a running system or against another profile.

Related tasks

Getting AIX Runtime Expert profile values
Use the **artexget** command to find information about a profile.

Applying AIX Runtime Expert profiles
To set a system with the configuration and tunable parameters from a profile, apply a profile using the **artexset** command.

Related information

artexget command

Modifying AIX Runtime Expert profiles

AIX Runtime Expert profiles are XML files and can be modified with any XML editor or any text editor.

User-created profiles using **artexget** command can be customized by changing the values of the parameters or by removing some of the parameters that are not required to modify or monitor the profile.

To modify AIX Runtime Expertprofiles, complete the following steps:

1. From the directory where `custom_all.xml` is located, run the following commands to save a copy of the profile:

```
cp custom_all.xml custom_all_backup.xml
```

2. From the directory where `custom_all.xml` is located, run the following command to edit the profile:

```
vi custom_all.xml
```

   Note: You can use any XML editor or text editor.
3. Modify the values of the parameters or remove the parameters that are not required to change or monitor the profile.
4. Run the following command to verify that the profile changes have been saved correctly by comparing them against the current system settings:

```
artexdiff -c -r custom_all.xml custom_all_backup.xml
```

   The **artexdiff** command displays the parameters that were modified by the editor. The <FirstValue> displays the value of the profile, and the <SecondValue> displays the value of the current system.

Related concepts

AIX Runtime Expert profiles

AIX Runtime Expert profiles are used to set values on a running system, extract values for a running system, and compare values against a running system or against another profile.

Related tasks

Getting AIX Runtime Expert profile values
Use the **artexget** command to find information about a profile.

Applying AIX Runtime Expert profiles
To set a system with the configuration and tunable parameters from a profile, apply a profile using the **artexset** command.

Related information

artexdiff command

Combining AIX Runtime Expert profiles

A profile can represent a full set of controls or any subset of controls. Another useful way to modify profiles is to combine profiles that represent a subset of controls using the **artxmerge** command.

You can use the **artexmerge** command to combine one or more profiles into a single profile.

To combine profiles complete, the following steps:

1. From the directory where the profiles are stored run the following command:

   ```
   artexmerge profile_name1.xml profile_name2.xml > new_profile_name.xml
   ```

2. Run the following command to view the profile and verify that it is a valid profile:

   ```
   artexget new_profile_name.xml
   ```

   Note: If the profiles you are combining have duplicate parameters, the process of combining the profiles will fail. Alternatively, if you use the **-f** flag, then the parameter values from the latest profile are used.

Related information

artexmerge command

Finding AIX Runtime Expert profiles

Use the **artexlist** command to find profiles in a given path and from an LDAP server.

To find profiles, complete the following steps:

1. If the profile is on the local system, run the following command:

   ```
   artexlist
   ```

2. If the profile is located on an LDAP server, run the following command:

   ```
   artexlist -l
   ```

By default, the command lists the profiles in the `/etc/security/artex/samples` directory. To override the default path with an environment variable, set the **ARTEX_PROFILE_PATH** to one or more semicolon delimited paths, or a path that can be passed as an argument.

Related information

artexlist command

Getting AIX Runtime Expert profile values

Use the **artexget** command to find information about a profile.

Using a profile, you can display the values from the profile or from the system in different formats (XML, CSV, or text) with different filters, such as parameters that need a reboot to take effect and parameters that need some services to be stopped and restarted.

Getting values from the system is useful in the following situations:

To take a snapshot of a system
When a system is configured correctly, you can save the configuration of the system by taking a snapshot. You can use this snapshot at a later date, if any of the parameters are changed and you do not remember which parameters were changed. The snapshot profile can be used to bring the system back to the desired configuration.

To clone the configuration of a system for use on other systems
After a system is configured and tuned in an environment, you can extract the system settings into an AIX Runtime Expert profile and apply the profile on other systems.

To debug a problem
When a problem is found on a production system, you can use a profile to set up the same system settings on a test system and than debug the problems on the test system.

To get information about a profile, complete the following steps:

1. Go to the directory where the profile you want to get information about is located.
2. To get information about the profile run the following command:

```
artexget name_of_profile.xml
```

Limitation: When a system has many users defined, the AIX Runtime Expert commands **artexget**, **artexset**, and **artexdiff** applied to profiles such as, chuserProfile.xml, coreProfile.xml, or all.xml profiles, requires more time to complete than usual.

Related concepts
AIX Runtime Expert profiles
AIX Runtime Expert profiles are used to set values on a running system, extract values for a running system, and compare values against a running system or against another profile.

Related tasks
Creating AIX Runtime Expert profiles
Use existing samples in the `/etc/security/artex/samples` directory to create a new profile with the **artexget** command. The sample profiles are a template for you to create a profile that you can modify and save into a custom file.

Modifying AIX Runtime Expert profiles
AIX Runtime Expert profiles are XML files and can be modified with any XML editor or any text editor.

Related information
artexget command

Applying AIX Runtime Expert profiles
To set a system with the configuration and tunable parameters from a profile, apply a profile using the **artexset** command.

To apply a user-created profile, complete the following steps:

1. Go to the directory where the profile you want to apply is stored.
2. To apply the profile to the system, run the following command:

```
artexset -c name_of_profile.xml
```

3. Optional: If you want to apply a profile every time the system restarts to maintain a consistent configuration, run the following command:

```
artexset -b name_of_profile.xml
```

Note: The restricted parameters are supported as read-only parameters. Therefore, the values of these parameters can be retrieved with the**artexget** command, but cannot be set with the **artexset** command.

Related concepts
AIX Runtime Expert profiles

AIX Runtime Expert profiles are used to set values on a running system, extract values for a running system, and compare values against a running system or against another profile.

Related tasks

Creating AIX Runtime Expert profiles
Use existing samples in the `/etc/security/artex/samples` directory to create a new profile with the **artexget** command. The sample profiles are a template for you to create a profile that you can modify and save into a custom file.

Modifying AIX Runtime Expert profiles
AIX Runtime Expert profiles are XML files and can be modified with any XML editor or any text editor.

Related information

artexset command

Rolling back AIX Runtime Expert profiles

Use the **artexset -u** command to reset the configuration settings to the previous configuration setting of a system. You can apply the system settings that were being used before the profile was applied.

You cannot use the **rollback** command if you have not changed the system settings during your current session.

Rolling back is not considered a re-imaging of an operating system. When you use the **rollback** command, you are not deleting or creating resources, you are not deleting or creating resources but instead reverting the runtime configuration values to the system's previous settings. With the **rollback** command you cannot roll back to settings from a particular time or date. You can only roll back to the systems previous settings before you made a change.

The **rollback** command can be used in the following cases:

- Testing configuration changes to a system. If the new configuration works poorly, you can quickly revert to a previously trusted configuration.
- Debugging a system. If a system starts running poorly, a roll back could confirm if configuration changes have played a part in a newly detected problem.
- Implementing a new profile in order to satisfy some special exception situation. For example, a specified action only occurs once a month on the system, and after it has been applied to your system you want to restore the system to its previous configuration.

To roll back to the previous system settings, complete the following steps:

1. To roll back a profile, run the following command:

   ```
   artexset -u
   ```

2. To verify that the roll back completed correctly, run the following command to compare system settings:

   ```
   artexdiff -f txt -r -profile_name.xml
   ```

   Note: The *profile_name.xml* is the name of the latest applied profile to the system.

   The differences between the system and the profile are displayed.

Related information

artexget command
artexlist command

Comparing AIX Runtime Expert profiles

Use the **artexdiff** command to compare two profiles or a profile values with system values.

To compare the profiles for two different systems complete the following steps:

1. Run the following command from system 1:

```
artexget -p all.xml > all_system1.xml
```

2. Run the following command from system 2:

```
artexget -p all.xml > all_system2.xml
```

To verify any configuration parameters are changed on a system after a period of time, for example, if you go on vacation and want to verify any changes while you were gone, run the following commands:.

- After you return from vacation, run the following command:

```
$ artexget -p all.xml > all_before_vacation.xml
```

- To view any configuration changes that occurred during your vacation, run the following command:

```
$ artexdiff -c -p all_before_vacation.xml
```

Related information
artexget command
artexlist command

Writing AIX Runtime Expert profiles

You can expand the scope of AIX Runtime Expert by adding catalogs and profiles that the program can use. You must be familiar with AIX Runtime Expert concepts before attempting to write new catalogs.

The smallest piece of information handled by AIX Runtime Expert is a parameter. Parameters can be tunable, configuration files, environment variables, properties of objects such as users, devices or subsystems (such objects are called targets in the AIX Runtime Expert context).

Parameters are aggregated into profiles according to the domain of activity (such as user, tcpi). Profiles are the intended means of interaction between the users and the AIX Runtime Expert framework. Profiles are input to the **artexget** command that retrieve the parameter value on the system and return a profile. Profiles (including values) are input to the **artexset** command that set the parameters to the value read into the profile.

Concepts in writing AIX Runtime Expert profile

AIX Runtime Expert profiles are XML files that contain a list of configuration parameters and optionally the parameter values and the usage flags.

Profiles can be located on the system being tuned when using the AIX Runtime Expert commands directly on the command line.

Profile locations
AIX Runtime Expert sample profiles are located in the directory `/etc/security/artex/samples`.

When you are writing a new catalog for the AIX Runtime Expert to support, it is recommended that you also write a sample profile that can be used as an entry for the **artexget** command. A sample profile is a read-only profile with no values assigned to the parameters. Existing sample profiles are located in the `/etc/security/artex/samples` directory. By default, the **artexlist** command lists only the profiles located in the default directory, but the default directory can be modified by setting the **ARTEX_PROFILE_PATH** environment variable. Multiple directories can be specified in this environment variable by using the : separator.

All profiles from the samples directory are merged during the installation of the **artex.base.samples** fileset, to form the **default.xml** profile that is used by the **snap** command. A profile that should not be part of the **default.xml** profile should not be delivered in the samples directory. Example of profiles that should not be included in the **default.xml** profile are the profiles that have the potential to include thousands of parameters (for example if it uses users as a target class) and profiles that should be run only on specific systems (for instance the **vios** attributes profile).

Profile naming
AIX Runtime Expert profiles are named based on the commands.

Profiles are usually built around a single command or a set of commands. Profiles may include several catalogs if the catalogs are closely related. The convention is to name the files based on a command say, **commandProfile.xml** for the sample profile and **commandParam.xml** for the catalog, but this is not mandatory. Only the**.xml** file extension is required.

Profile process
Discusses the process to write a new AIX Runtime Expert profile.

The following steps are required to be performed when writing a new AIX Runtime Expert profile:

1. Make a list of the parameters you want in the profile.
2. Create an **<Parameter name="...">** element for each of the parameters, setting the *name* attribute to the name used in the catalog file **<ParameterDef>** element.
3. Group all parameters defined in a same catalog file inside the same **<Catalog id="...">** element, setting the *id* attribute to the same id used in the catalog file **<Catalog>** element.
4. For each **<Parameter>** element, do the following:
   a. If the parameter is defined with the *reboot*=`true` in the catalog file, add the *reboot*=`true` and *applyType*=`nextboot` attributes.
   b. If the parameter must be only captured and not set, add the `readOnly=true` attribute.
   c. If the parameter is defined with a non-empty *targetClass* attribute in the catalog file, do the following:
      i) If target discovery is desired for this parameter, then define a single **<Parameter>** element for this parameter and use the special **<Target class="" instance="" >** target for this element.
      ii) If specific targets need to be defined for this parameter, then define one **<Parameter>** element for each target. Under each **<Parameter>** element, define the appropriate **<Target class="..." instance="..." />** elements to specify the target completely.
5. Test the profile by running the **artexget –r** command.

AIX Runtime Expert profile elements

<Profile> element
The **<Profile>** element is the root element for all profile files.

Syntax

The following attributes are supported:

Table 2. Attributes

Attribute	Required	Type	Description
origin	no	string	Origin of the profile.
date	no	dateTime	Date of creation or last modification of the profile. Format is YYYY-MM-DDThh:mm:ss.
readOnly	no	boolean	Tells whether this profile can be used in a set operation. Default value is`false`.
version	no	string	Version number of the profile.

The following child elements are supported:

Table 3. Child Elements

Child element	Required	Number	Description
<ShortDescription>	no	0 - 1	Short textual description of the catalog.
<Description>	no	0 – 1	Long textual description of the catalog.
<Comments>	no	0 – 1	User-provided comments.
<Catalog>	no	0 – any	Catalog required to handle the operations on a profile.

Attributes

The *origin* attribute

The *origin* attribute is an informational attribute that can be assigned the following values:

- When creating a sample profile, the *origin* attribute must be set to `reference`.
- When a profile is created by using the **artexget** command, the *origin* attribute is automatically set to `get`.

Child elements

The **<Comments>** element is an optional string reserved for other purposes. This element must not be used when a profile is created manually, and it not used by the base AIX Runtime Expert commands.

Examples

1. An empty sample profile would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile origin="reference" version="2.0.0" readOnly="true">
</Profile>
```

2. The **artexget -r /etc/security/artex/samples/smtctmProfile.xml** command outputs a profile similar to following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile origin="get" version="2.0.1" date="2010-09-29T07:50:56Z">
  <Catalog id="smtctlParam" version="2.0">
    <Parameter name="enableSMT" value="1"/>
  </Catalog>
</Profile>
```

Related information

The <Catalog> element

The <Description> and <ShortDescription> element.

<Description> and <ShortDescription> elements
The **<Description>** and the **<ShortDescription>** elements can be used to provide a textual description for profiles and parameters.

Syntax

The parent element of the **<ShortDescription>** element is:

- **<Profile>** element

The parent element of the **<Description>** element can be:

- **<Profile>** element
- **<Parameter>** element

Both the **<Description>** and **<ShortDescription>** elements have the same format. The text contained by the **<Description>** element is the string content of the XML tag.

Usage

Descriptions in profile files are not currently used by the AIX Runtime Expert framework. AIX Runtime Expert commands ignore any comments included in the input profile.

Examples

Following is an example of the **<Description>** and **<ShortDescription>** elements:

```
<ShortDescription>
   Short summary of field contents.
</ShortDescription>
<Description>
This text field can be used to display in full detail the use of the parent element.
</Description>
```

Related information

The <Profile> element.

The <Parameter> element.

<Catalog> element
The **<Catalog>** element indicates the name of the catalog file that contains the definitions for the child **<Parameter>** elements.

Syntax

Parent element: **<Profile>**

The following attributes are supported:

Table 4. Attributes

Attribute	Required	Type	Description
id	yes	string	Specifies the catalog id. This name should be unique on the system.
version	no	string	Specifies the version of the catalog used to build this profile.

The following child elements are supported:

Table 5. Child elements			
Child element	**Required**	**Number**	**Description**
<Parameter>	no	0 – any	Parameter included in the catalog.
<SubCat>	no	0 – any	Subcategory included in the catalog.
<Seed>	no	0 – any	Seed included in the catalog.

Attributes

id attribute

The *id* attribute must be set to the name of the catalog that defines the parameters listed under the **<Catalog>** element. The *id* attribute is the base name of the catalog file on the disk, with the **.xml** extension removed. For example, a profile will use the **<Catalog id="commandParam">** element to reference the `commandParam.xml` catalog file.

By default, the catalog files are searched under the `/etc/security/artex/catalogs` directory. However it is possible, for the root user only, to search other directories by setting the **ARTEX_CATALOG_PATH** environment variable. Multiple directories can be specified in this environment variable using the : separator.

version attribute

The *version* attribute is written as *MM.mm* where *MM* is the major number and *mm* is the minor number.

The *version* attribute must match the version of the referenced catalog file (see The <Catalog> element in the section Writing AIX Runtime Expert catalogs). If an AIX Runtime Expert command is run on a profile that references a catalog with the incorrect version, the following warning message is displayed:

```
0590-218 Catalog version differs from the one referenced in profile
Local catalog version is '2.1'. Version used to build profile was '2.0'
```

Usage

The **<Catalog>** element identifies the catalog file that contains the definition of the listed seeds and parameters. All seeds and parameter elements in a profile must be located in the appropriate **<Catalog>** element.

A profile may reference several catalogs. For example, the **default.xml** profile is built during the installation of the `artex.base.sample` fileset by merging a select set of other sample profiles.

Examples

The security attributes profile `secattrProfile.xml` uses three catalogs, each catalog handling, one of the security tables:

```
<Profile origin="reference" readOnly="true" version="2.0.0">
  <Catalog id="privcmdParam" version="2.0"
    <Parameter name="privatecommands" />
  </Catalog>
  <Catalog id="privdevParam" version="2.0">
    <Parameter name="privatedevices"/>
  </Catalog>
  <Catalog id="privfileParam" version="2.0">
    <Parameter name="privatefiles" />
  </Catalog>
</Profile>
```

Related information

The **<Catalog>** element (in catalog files).

<SubCat> element

The **<SubCat>** element provides a means to create logical subcategories under a **<Catalog>** element.

Syntax

Parent element: **<Catalog>**, **<SubCat>**

The following attributes are supported:

Table 6. Attributes

Attribute	Required	Type	Description
id	yes	string	Specifies the subcategory name. This name should be unique within a same **<Catalog>** element.

The following child elements are supported

Table 7. Child Elements

Child element	Required	Number	Description
<Parameter>	no	0 – any	Contains a parameter name.
<SubCat>	no	0 – any	Nested subcategory

Attributes

The *id* attribute uniquely identifies a subcategory within a catalog. A profile may include several subcategories with the same id, provided that these are not used under the same **<Catalog>** element.

Child elements

A **<SubCat>** element may have another **<SubCat>** as a child element. There is no limit to the number of nested subcategories you can define.

Usage

Subcategories are only included for readability. They do not affect the way parameters are handled.

Examples

The `noProfile.xml` profile includes several subcategories. Following is an example:

```
<Profile origin="reference" readOnly="true" version="2.0.0">
  <Catalog id="noParam" version="2.0">
    <SubCat id="general_network"
      <Parameter name="fasttimo"/>
      <Parameter name="nbc_limit"/>
    </SubCat>
    <SubCat id="tcp_network">
      <Parameter name="clean_partial_conns"/>
      <Parameter name="delayack"/>
    </SubCat>
    <SubCat id="restricted">
      <Parameter name="extendednetstats" readOnly="true"/>
      <Parameter name="inet_stack_size" readOnly="true"/>
```

```
    </SubCat>
  </Catalog>
</Profile>
```

Related information

The <Parameter> element.

<Parameter> element
The **<Parameter>** element defines a configuration parameter.

Syntax

The following attributes are supported:

Table 8. Attributes

Attribute	Required	Type	Description
name	yes	string	Specifies the name of the parameter. This name must be unique within a catalog.
value	no	string	Value of the parameter, if it is defined.
applyType	no	string	Specifies whether the runtime or next boot value of the parameter must be retrieved or set if no flag is specified. Default value is `runtime`.
reboot	no	boolean	When `true`, indicates that a reboot is required before a value change is effective. Default value is `false`.
readOnly	no	boolean	Specifies whether the value of the parameter cannot be set. Default: value is `false`.
disruptive	no	boolean	Specifies whether the method used to set the parameter implies disruptive constraints. Default value is `false`.
setDiscover	no	boolean	Specifies whether the set method must be set to the value for all discovered instances of a target class. Default value is `false`.

The following child elements are supported

Table 9. Child Elements			
Child element	**Required**	**Number**	**Description**
<Value>	no	0 - 1	Value of the parameter.
<Target>	no	0 - any	Target to which the parameter applies.
<Description>	no	0 - 1	Description of the parameter.
<Property>	no	0 - any	Property of the parameter.

Attributes

Table 10. Attributes	
Attribute	**Description**
name	The name of the parameter is the only required attribute of the **<Parameter>** element. Along with the catalog name specified in the parent **<Catalog>** element, the parameter name uniquely identifies a parameter definition in the catalog file.
value	The value of the parameter can be specified either as an attribute or as a child element.
applyType	The *applyType* attribute can take the values `runtime` (the default value) or `nextboot`. This attribute determines the command that is used to handle the parameter. For set operations, *applyType*=`runtime` indicates that the **<Set type="permanent">** command from the catalog file should be used to set the parameter. The *applyType*=`nextboot` indicates that the **<Set type="nextboot">** command should be used instead. For get operations, when the *–p* flag is used, *applyType*=`runtime` indicates that the **<Get type="current">** command from the catalog file must be used to obtain the parameter. The *<applyType>*=`nextboot` indicates that the **<Get type="nextboot">** command should be used instead. The *applyType* attribute must be set to `nextboot` if the *reboot* attribute is set to true.

Table 10. Attributes (continued)	
Attribute	**Description**
reboot	This attribute has a default value of false. When set to true, it means that the system must be rebooted before any change to the parameter is effective. This attribute must match the *reboot* attribute defined in the corresponding **<ParameterDef>** element of the catalog file. When this attribute is set to true, the *applyType* attribute must be set to `nextboot`. When you set a parameter that has the *reboot* attribute set, a warning is displayed to the user, stating that a reboot operation is needed: `0590-206 A manual post-operation is required for the changes to take effect Please reboot the system` Only set the *reboot* attribute to true when a change to the parameter value will not be effective until after the next reboot.
readOnly	This attribute indicates that the value of the parameter can be read by the **artexget** command, but that it will not be set using the **artexset** command, nor taken into account in a comparison operation with live values by using the **artexdiff** commands. The default value is false. A few situations that can warrant setting the *readOnly* attribute to true follow: • The parameter is static and its value cannot be modified (e.g. the *memory_frames* parameter in the **vmo** command). • Access to the parameter is restricted and it is not recommended for the user to modify it in automatic procedures. In this case, the set configuration methods should be defined for this parameter in the catalog file, but a system administrator must remove manually the *readOnly* attribute from the profile to be able to set the parameter.
setDiscover	The *setDiscover* attribute, when set to true indicates that, when the **artexset** command is called with the *–d* flag, the **discover** command must be called to discover all instances of the target, and set all to the value stored in the profile. The *setDiscover* default value is false. A value of true only makes sense if the parameter has target classes defined in the catalog file. Do not specify this attribute when creating a sample profile. Advanced users must add this attribute manually when deemed necessary.

Other attributes

The *type* and *disruptive* attributes are informative attributes that are automatically set by the **artextget** command when it is called with the **-i** flag. Do not include these attributes when you are creating a sample profile.

Examples

1. The following example is an excerpt from the `vmoProfile.xml` sample catalog, showcasing the use of various optional attributes:

```
<Profile origin="reference" readOnly="true" version="2.0.0">
  <Catalog id="vmoParam" version="2.1">
    <Parameter name="nokilluid"/>
    <Parameter name="memory_frames" readOnly="true"/>
    <Parameter name="kernel_heap_psize" reboot="true" applyType="nextboot"/>
  </Catalog>
</Profile>
```

2. If you run the **artexget -r** command on the profile from example 1 the following profile is displayed:

```
<Profile origin="get" version="2.0.1" date="2011-03-24T13:41:01Z">
  <Catalog id="vmoParam" version="2.1">
    <Parameter name="nokilluid" value="0"/>
    <Parameter name="memory_frames" value="393216" readOnly="true"/>
    <Parameter name="kernel_heap_psize" value="4096" applyType="nextboot" reboot="true"/>
  </Catalog>
</Profile>
```

Related information

The Parameter values topic.

The <ParameterDef> element.

Parameter values
The value of a parameter can be set in a profile either as an attribute if they are short enough, or as a child element of the **<Parameter>** element.

Use

When writing a sample profile, no value must be assigned to the parameters. The value of a parameter, if it exists, is automatically included in the profile obtained by running an **artexget** command.

The runtime and nextboot values

The concept of `runtime` and `nextboot` values is an important part of the AIX Runtime Expert framework.

The runtime value of the parameter is its current value retrieved on the system at the time the **artexget** command is run. The `nextboot` value is the value the parameter will have after the next reboot of the system.

For example, the parameter *type_of_dump* in the profile `sysdumpdevProfile.xml`. The current (`runtime`) value of this parameter may be `traditional` or `firmware-assisted`. If this value is changed (using the **artexset** command or directly by using the **sysdumpdev** command), it will not be effective until the next reboot. The `nextboot` value of this parameter will then be the modified value.

```
<Parameter name="type_of_dump" applyType="nextboot" reboot="true" />
```

Example

The following example shows a parameter with the value specified as an attribute, and another parameter with the value specified as a child element:

```
<Profile origin="get" version="2.0.1" date="2010-09-28T12:30:03Z">
<Catalog id="login.cfgParam" version="2.0">
<Parameter name="shells">
<Value>
/bin/sh,/bin/bsh,/bin/csh,/bin/ksh,/bin/tsh,
/bin/ksh93,/usr/bin/sh,/usr/bin/bsh,/usr/bin/csh,
/usr/bin/ksh,/usr/bin/tsh,/usr/bin/ksh93,
/usr/bin/rksh,/usr/bin/rksh93,
/usr/sbin/uucp/uucico,/usr/sbin/sliplogin,
/usr/sbin/snappd
</Value>
</Parameter>
<Parameter name="maxlogins" value="32767"/>
</Catalog>
</Profile>
```

<Property> element
The **<Property>** element assigns a value to a parameter property.

Syntax

Parent element: **<Parameter>**

The following attributes are supported:

Table 11. Attributes

Attribute	Required	Type	Description
name	yes	string	Specifies the name of the property.
value	no	string	Specifies the value of the property.

Usage

The **<Property>** element assigns a value to the property name of the parent element. This value is used when the **%p***[name]* sequence is expanded during command-line generation.

The **<Property>** element is generally not added manually to profiles. The element is inserted automatically in the output profile when the **artexget -r** and **artexget -n** commands are run, based on the command defined under the corresponding **<Property>** element of the catalog file.

Example

The following example sets the **nodeId** property for the `netaddr` parameter. The property value is captured by the **artexget -r** command and is the output of the **uname -f** command:

```
<Parameter name="netaddr" value="172.16.128.13">
 <Target class="device" instance="en0"/>
 <Property name="nodeId" value="8000108390E00009"/>
</Parameter>
```

Related information

The artex_catalog_elem_PropertyDef.dita (in catalog files).

<Seed> element
The **<Seed>** element defines a seed that is expanded into one or more **<ParameterDef>** elements during the **<Get>** operation.

Syntax

Parent element: **<Catalog>**

The following attribute is supported:

Table 12. Attribute

Attribute	Required	Type	Description
name	yes	string	Specifies the name of the seed element that matches a **SeedDef** element in the catalog file.

The following child elements are supported:

Table 13. Child elements

Child element	Required	Number	Description
<Parameter>	no	0 – any	Filters discovered parameters based on the names of the parameters.
<Target>	no	0 – any	Filters discovered parameters based on their targets.

Usage

The **<Seed>** element discovers parameters dynamically during a **<Get>** operation.

When the **artexget** command is issued, each **<Seed>** element in the input profile is expanded into one or more **<Parameter>** elements. The profiles are expanded based on the rules defined in the matching **<SeedDef>** element of the catalog file. This process is called parameter discovery. After the parameter discovery process is completed, the **artexget** command proceeds as usual with the expanded profile.

The optional **<Parameter>** and **<Target>** child elements are used to filter the discovered parameters. Discovered parameters that do not match the criteria defined in the **<Parameter>** subelement, are discarded. Also, those parameters that apply to targets that do not match the criteria defined in the **<Target>** subelement, are discarded.

Examples

This example uses the **devSeed** catalog to define a seed that can be used to discover all attributes of all devices:

```
<?xml version="1.0" encoding="UTF-8"?>
<Catalog id="devSeed" version="3.0">
  <SeedDef name="devAttr">
    <Discover>
      <Command>
        /usr/sbin/lsdev -F 'name class subclass type' |
        while read DEV CLASS SUBCLASS TYPE
        do
          CAT=devParam.$CLASS.$SUBCLASS.$TYPE
          /usr/sbin/lsattr -F attribute -l $DEV |
```

```
          while read PAR
          do
            echo "device=$DEV $CAT $PAR"
          done
        done
      </Command>
      Mask target="1" catalog="2" name="3">(.*) (.*) (.*)</Mask>
    </Discover>
  </SeedDef>
</Catalog>
```

The following profile can be used to discover all the supported attributes of all the supported devices:

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile>
  <Catalog id="devSeed" version="3.0">
    <Seed name="devAttr"/>
  </Catalog>
</Profile>
```

2. Using the same catalog, a **<Target>** filter can be used to discover all the supported attributes of all Ethernet adapters:

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile>
  <Catalog id="devSeed" version="3.0">
    <Seed name="devAttr">
      <Target class="device" match="^en[0-9]+$"/>
    </Seed>
  </Catalog>
</Profile>
```

3. A **<Parameter>** filter can be added to capture only the **netaddr**, **netaddr6**, **alias**, and **alias6** attributes of all Ethernet adapters:

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile>
  <Catalog id="devSeed" version="3.0">
    <Seed name="devattr">
      <Parameter match="^(netaddr|alias)6?$"/>
      <Target class="device" match="^en[0-9]+$"/>
    </Seed>
  </Catalog>
</Profile>
```

Related information

The artex_catalog_elem_SeedDef.dita element (in catalog files).

<Target> element
A **<Target>** element defines the instance of a target class to which the parameter applies.

Syntax

Parent element: **<Parameter>**

Multiple occurrences of the same parameter from the same catalogs are allowed if, and only if, they apply to different instances of their target.

The following attributes are supported:

Table 14. Attributes

Attribute	Required	Type	Description
class	yes	string	Specifies the name of the target class.
instance	no*	string	Specifies the name of the instance of a class.

Table 14. Attributes (continued)			
Attribute	**Required**	**Type**	**Description**
match	no*	string	Specifies the regular expression as applied to the discovered instance names.

* One and only one of the *instance* and *match* attributes must be specified.

Use

Some parameters do not apply to the system as a whole, but to a specific object. An example is the home directory of a user as specified in the `chuserProfile.xml` profile; this parameter applies to a specific user (root, guest) in a specific loadable module (files, LDAP). In this example, the user and the module are two target classes. The *home* parameter applies to specific instances of these target class. For example, the guest instance of the user class, and the files instance of the module class.

If the *class* and *instance* attributes are both set to the empty string, then a discovery is performed for this parameter when the **artexget** command is run on such a profile, the discovery method declared in the corresponding catalog file is executed, and a parameter is created in the output profile for each discovered instance of the parameter. See example 1.

If the *class* and *instance* attributes are both specified, then the target is fully qualified and the parameter only applies to the specified instance of the target class. See example 2.

If the *class* and *match* attributes are both specified, a discovery is performed as above, but only target instances with a name that matches the regular expression specified in the *match* attribute are discovered. See example 3.

When writing a sample profile, the *class* and *instance* attributes must be left empty. This means that, when encountering the empty target class, the **artexget** command will discover the list of the instances of that target class (all the users or the subsystems on the system) before retrieving the values.

Running the **artexset** command on an undiscovered target class displays a warning:

```
0590-216 Some parameters in the profile require a target discovery and will be ignored
```

Examples

1. An example of a profile with targets before discovery is the `chuserProfile.xml` profile that defines the home directory of a user. The following example shows a sample profile:

```
<Profile version="2.0.0" origin="reference" readOnly="true">
  <Catalog id="chuserParam" version="2.0">
    <Parameter name="home">
      <Target class="" instance=""/>
    </Parameter>
  </Catalog>
</Profile>
```

2. After discovery, the `chuserProfile.xml` profile would contain a copy of the home parameter for each discovered user in each of the discovered loadable modules:

```
<Profile version="2.0.0" origin="get">
  <Catalog id="chuserParam" version="2.0">

    <Parameter name="home" value="/">
      <Target class="user" instance="root"/>
      <Target class="module" instance="files"/>
    </Parameter>

    <Parameter name="home" value="/etc">
      <Target class="user" instance="daemon"/>
      <Target class="module" instance="files"/>
```

```
    </Parameter>

    ...

  </Catalog>
</Profile>
```

3. The following profile uses the *match* attribute to discover the home directory of all users with a name that starts with a *u* in the file module:

```
<Profile version="2.0.0" origin="reference" readOnly="true"
  <Catalog id="chuserParam" version="2.0">
    <Parameter name="home">
      <Target class="user" match="^u"/>
      <Target class="module" instance="files"/>
    </Parameter>
</Catalog>
</Profile>
```

Related information

The **<Discover>** element (in catalog files).

Writing AIX Runtime Expert catalogs

Catalog files are used internally by the AIX Runtime Expert framework.

The catalog files contain the parameter definitions and binding information to configuration methods that describe the commands used to retrieve or set parameter values. Catalog files are local to the system which is being tuned and configured.

AIX Runtime Expert catalog concepts

Catalog files contain all the information required to perform operations on parameters, including definitions, conditions of use, and configuration methods. Catalog files must not be manipulated directly by users and are only used through the AIX Runtime Expert core engine.

Catalogs are installed on a system at the same time as the AIX Runtime Expert core engine. When new catalogs are linked to components or third-party applications that are installed on a system, it is important to ensure they are in level with the installed AIX Runtime Expert core engine.

Catalog location
AIX Runtime Expert catalog files are stored in the `/etc/security/artex/catalogs` directory.

The name of a catalog file must exactly match its *id* attribute, suffixed with `.xml` extension. For example, a catalog named `commandParam.xml` must have an *id* attribute value `commandParam`.

In order to be located by the profile that reference it, the catalog must have the same name in the catalog XML file and in the **<Catalog>** element of the profile XML file. By default, the AIX Runtime Expert core engine looks for catalogs in the default directory `/etc/security/artex/catalogs`. This behavior can be changed, for the root user only, by setting the **ARTEX_CATALOG_PATH** environment variable. Multiple directories can be specified in this environment variable using the : separator.

Catalog process
Steps to write a new AIX Runtime Expert catalog.

The following steps are required to be performed when writing a new AIX Runtime Expert catalog:

1. Make a list of parameters you want in the catalog file.
2. For each parameter, create a **<ParameterDef>** element
3. If several parameters use the same command for a **<Get>**, **<Set>**, **<Discover>** or **<Diff>** operation:
   - Define a **<CfgMethod>** element at the top of the catalog.
   - Use the *cfgmethod* attribute to inherit from the configuration method.

4. If several parameters are subject to the same constraint, define a **<ConstraintDef>** element at the top of the catalog.
5. For each parameter:
   a. Define the **<Get type="current">** and **<Get type="nextboot">** operations for each parameter, either directly under the **<ParameterDef>** element, or referenced under the **<CfgMethod>** element, or using any of the combinations.
   b. Define all the supported **<Set>** operations for each parameter, either directly under the **<ParameterDef>** element, or under the referenced **<CfgMethod>** element, or using any combination of those possibilities.
   c. If the parameter requires a target:
      i) Define the supported target classes using the *targetClass* attribute
      ii) Define the discover operation, either directly under the **<ParameterDef>** element, or the referenced **<CfgMethod>** element, or using any combination of those possibilities. In most cases the discover method will be defined in a configuration method.
   d. If the parameter requires a reboot for a change to take effect, add the *reboot* =`true` attribute.
   e. If the parameter is subject to a constraint, either define a **<ConstraintDef>** element under the **<ParameterDef>** element, or use the constraint attribute to reference an existing constraint.
6. To test the catalog file:
   a. Create a profile with all the parameters defined in the catalog file.
   b. Use the **`artexget -r`** command to capture values and test the **<Discover>** and **<Get>** operations.
   c. Use the **`artexset -c -F -R -l all`** command on the resulting profile to test the **<Set>** and **<Diff>** operations.
   d. Additionally, the **`-g 3 -g COMMANDS`** flags can be added to those two commands to get more information about the command line generated to perform the requested operation.

Related information

See the topic about the **<Catalog>** root element.

AIX Runtime Expert catalog elements

<Catalog> element
The **<Catalog>** element is the root element for all catalog files.

Syntax

The following attributes are supported:

Table 15. Attributes

Attribute	Required	Type	Description
id	yes	string	Specifies the name of the catalog. This name must be unique on the system.
version	no	string	Specifies the version number of the catalog.
date	no	dateTime	Specifies the date of creation. Format is *YYYY-MM-DDThh:mm:ss*.

Table 15. Attributes (continued)			
Attribute	**Required**	**Type**	**Description**
priority	no	integer	Specifies the order of execution of the catalog relative to others in the set methods. Default value is 0.
inherit	no	string	Specifies the name of a catalog to inherit.

The following child elements are supported. The *number* column defines how many occurrences of the child are allowed:

Table 16. Child elements			
Child element	**Required**	**Number**	**Description**
<ShortDescription>	no	0 – 1	Short textual description for the catalog.
<Description>	no	0 – 1	Long textual description for the catalog.
<SubCat>	no	0 – any	Sub category
<ParameterDef>	no	0 – any	Contains the properties of a parameter.
<ConstraintDef>	no	0 – any	Parameter constraints definition (conditions and disruptive commands).
<CfgMethod>	no	0 – any	Configuration method definition
<PrereqDef>	no	0 – any	Defines a prerequisite.
<PropertyDef>	no	0 – any	Defines a property.
<SeedDef>	no	0 – any	Defines a seed.

Attributes

Table 17. Attributes	
Attribute	**Description**
id	The *id* attribute should match the catalog file name, stripped from its `.xml` extension. The catalog id is referenced in profiles using the **<Catalog>** element.

Table 17. Attributes (continued)	
Attribute	**Description**
Priority	The *priority* attribute is used when the set methods of a specific catalog are required to be run before or after the set methods of other catalogs when they are included in the same profile (for example, the compound profile **default.xml**). The default priority of a catalog is 0. The rule is that when two catalogs share the same priority, their set methods are run in an undefined order. If a catalog has a priority with a positive value, its set methods are executed before the others, in the descending priority order. If a catalog has a priority with a negative value, its set methods are executed after the others, in the descending priority order.
Version	The *Version* attribute is present in both profiles and catalogs. The version helps identifying whether profiles and catalogs are compatible with the AIX Runtime Expert core engine and with each other. See *Version* attribute for more details.
Date	The *date* attribute is not currently used for the **<Catalog>** element. It is included for future use and maintainability.
inherit	The *inherit* attribute specifies the name of a catalog to inherit from, without the .xml extension. All the elements defined in the inherited catalog are available in the main catalog, as if they were defined locally.

Example

Following is an example of a catalog using the *priority* attribute. The `aixpertParam.xml` catalog sets security options and must be set after all other catalogs have been set. Thus, its priority is set to a high negative value.

```
<Catalog id="aixpertParam" version="2.0" priority="-1000">
```

Related information

The <ConstraintDef> element.

The <CfgMethod> element.

The <Description> and <ShortDescription> elements.

The <ParameterDef> element.

The <SubCat> element.

Version attribute

Syntax

The version of a catalog is written as an attribute in the format *MM.mm* where *MM* is the major number and *mm* is the minor number.

```
<Catalog id="commandParam" version="2.0">
```

Major version number

The major version number is the same for all AIX Runtime Expert catalogs installed on a system, and the whole AIX Runtime Expert framework, where it is referenced. This major number is increased at each major change of the XML schema of the profiles and catalogs.

When creating a new catalog, set the major version number to the current AIX Runtime Expert core engine version number, which can be found by looking inside any of the standard catalog files that come with the `artex.base.rte` fileset.

If an **artexget** command is invoked on a profile whose major version number differs from the one referenced in the AIX Runtime Expert core engine, the command fails with the following error:

```
0590-117 Version error
This profile was created on a version unsupported by ARTEX
```

It is also advised that a profile and a catalog share the same major version number in order to be compatible. A profile references catalogs with a specific version number. If the major version number of the profile is not the same as the catalog major version number, any AIX Runtime Expert command will display a warning to inform the user that results may be unpredictable:

```
0590-218 Catalog version differs from the one referenced in profile
```

Minor version number

The minor version number is specific to each catalog and is increased each time a major change in the catalog makes it incompatible with the previous version. A profile references catalogs with a specific version number. If the profile minor version number is not the same as the catalog minor version number, any AIX Runtime Expert command will issue a warning to inform the user that results may be unpredictable:

```
0590-218 Catalog version differs from the one referenced in profile
```

When creating a new sample profile or catalog, set the minor version number to 0.

<Description> and <ShortDescription> elements
Descriptions are optional informative text fields that can be added to various elements in the catalog files. These fields are optional, but it is recommended that catalog writers use them to document the parent element.

Syntax

The parent element of a **<ShortDescription>** element can be one of:

- **<Catalog>**
- **<SubCat>**

The parent element of a **<Description>** element can be one of:

- **<Catalog>**
- **<SubCat>**
- **<ParameterDef>**
- **<ConstraintDef>**

The content of the **<Description>** and **<ShortDescription>** elements is either a simple string, or a translated message defined by one of the **<NLSCatalog>**, **<NLSSmitHelp>** or **<NLSCommand>** elements. See the Globalization support topic for more information.

Usage

Currently, only the description of **<ParameterDef>** elements is retrieved and displayed by the **artexget** command with the **-i** flag. It is recommended to provide globalization for the text included in those description fields.

The description field of the other elements are currently not used by the AIX Runtime Expert framework, but they should be provided for future use and for documentation purpose.

Example

1. Here is a simple example of description fields:

```
<ShortDescription>
   chuser parameters
</ShortDescription>
<Description>
   Parameter definition for the chuser command
</Description>
```

2. The same example, using translated messages from the `artexcat.cat` message file:

```
<ShortDescription>
<NLSCatalog catalog="artexcat.cat" setNum="12" msgNum="1">
  chuser parameters
</NLSCatalog>
</ShortDescription>
<Description>
<NLSCatalog catalog="artexcat.cat" setNum="12" msgNum="2">
  Parameter definition for the chuser command
</NLSCatalog>
</Description>
```

Globalization support
This section describes how globalization is implemented in the descriptive fields of the AIX Runtime Expert catalogs.

Syntax

Parent element: **<Description>**, **<ShortDescription>**

The parent element may contain one (and only one) of the following child elements:

Table 18. Child Elements

Child element	Required	Number	Description
<NLSCatalog>	no	0 – 1	String included in a message catalog
<NLSSmitHelp>	no	0 – 1	String included in a SMIT help HTML file
<NLSCommand>	no	0 – 1	String issued by an AIX command

NLS Catalog

The NLS Catalog globalization format is used when the localized message to display is included in an existing message catalog in the **catgets()** format.

The **<NLSCatalog>** element contains the following attributes:

Table 19. Attributes

Attribute	Required	Type	Description
catalog	yes	string	Name of the catalog where the message resides
setNum	yes	integer	Number of the message set where the message resides

Table 19. Attributes (continued)			
Attribute	**Required**	**Type**	**Description**
msgNum	yes	integer	Number of the message in the message set

If the localized message catalog does not exist, the default message is displayed instead. The default message is, optionally, included as the contents of the **<NLSCatalog>** element. It is a recommended practice to provide a default message.

NLS SMIT Help

The NLS Smit Help globalization format is used when the localized message to display already exists in a SMIT help HTML file.

The **<NLSSmitHelp>** element contains the following attribute:

Table 20. Attributes			
Attribute	**Required**	**Type**	**Decsription**
msgId	yes	integer	The **help_msg_id** field provided in the SMIT stanza

If the localized help file does not exist, the default message is displayed instead. The default message is, optionally, included as the contents of the **<NLSSmitHelp>** element. It is a recommended practice to provide a default message.

NLS Command

The NLS Command globalization format is used when the localized message to display is issued by an AIX command. This is the case for all tuning commands (like **no**, **vmo**) that provide a *–h* flag to display help text for a specific parameter.

The **<NLSCommand>** element contains the following attribute:

Table 21. Attribute			
Attribute	**Required**	**Type**	**Description**
command	command	string	Shell expression to execute

Examples

1. Example of the **<NLSCatalog>** element from the `chssysParam.xml` AIX Runtime Expert catalog, including a default message.

```
<Description>
  <NLSCatalog catalog="artexcat.cat" setNum="10" msgNum="2">
    Changes a subsystem definition in the subsystem object class.
  </NLSCatalog>
</Description>
```

2. Example of the **<NLSSmitHelp>** element:

```
<Description>
  <NLSSmitHelp msgId="055136"/>
</Description>
```

3. Example of the **<NLSCommand>** element from the `schedoParam.xml` catalog:

```
<Description>
  <NLSCommand command="/usr/sbin/schedo -h maxspin | /usr/bin/tail -n +2"/>
</Description>
```

<SubCat> element
Subcategories, optional parameters, subsets inside a catalog, can be specified by using the **<SubCat>** element inside a catalog file.

Syntax

Parent element: **<Catalog>**,**<SubCat>**

The following attributes are supported:

Table 22. Attributes

Attribute	Required	Type	Description
id	yes	string	Specifies the name of the catalog subcategory. This name should be unique per catalog file.

The following child elements are supported:

Table 23. Child Elements

Child element	Required	Description
<ShortDescription>	no	Short textual description of the subcategory.
<Description>	no	Long textual description of the subcategory.
<SubCat>	no	Nested subcategory. This element may occur multiple times.
<ParameterDef>	no	Contains the properties of a parameter. This element may occur multiple times.

Attribute

A subcategory is local to a catalog:

- A subcategory id is unique inside a catalog file.
- Multiple catalogs can make use of the same subcategory identifier.

The subcategories defined in a catalog must exactly match the subcategories reported in the associated sample profile.

Related information

The <Description> and <ShortDescription> elements.

The <SubCat> element.

The <ParameterDef> element.

<ParameterDef> element
AIX Runtime Expert are defined in a catalog file by using the **<ParameterDef>** element.

Syntax

Parent element: **<Catalog>**,**<ParameterDef>**

The following attributes are supported:

Table 24. Attributes

Attribute	Required	Type	Description
name	yes	string	Specifies the name of the parameter. This name must be unique per catalog.
type	yes	string	Specifies the parameter type, as seen from the core engine.
targetClass	no	string	Specifies the target classes for the parameter if any.
reboot	no	boolean	When true, indicates that a reboot is required. Default value is false.
cfgmethod	no	string	Specifies the *id* of the configuration method defined at **<Catalog>** level that contains methods to use for this parameter.
constraint	no	string	Specifies the id of a constraint defined at <Catalog> element level for the current catalog file.
priority	no	integer	Execution rank of this parameter in the set method relative to other parameters in this catalog. Default value is 0.

The following child elements are supported:

Table 25. Child Elements

Child element	Required	Description
<Description>	no	Textual description of the parameter.
<ConstraintDef>	no	Parameter constraint definition (disruptive commands).

Table 25. Child Elements (continued)		
Child element	**Required**	**Description**
<Get>	no	Configuration method definition for the get operation. This element may occur multiple times.
<Set>	no	Configuration method definition for the set operation. This element may occur multiple times.
<Diff>	no	Configuration method definition for **`diff`** operation.
<Discover>	no	Configuration method definition for target discovery.

Attribute

Table 26. Attributes	
Attribute	**Description**
name	The name attribute uniquely identifies a parameter within a catalog file. See the parameter name attribute topic for more information.
type	The required type attribute indicates the value type of the parameter. The supported values are: • string, for alphanumerical strings; • integer, for numerical values; • integer-bi, for numerical values with an optional uppercase or lowercase K, M, G, T, P or E suffix for “kilo”, “mega”, “giga”, “tera”, “peta” and “exa”. These suffixes are interpreted as powers of 1024; • integer-si, for numerical values with an optional SI suffix. Same as the integer-bi type, except suffixes are interpreted as powers of 1000. • boolean, for boolean values. Supported values are 0 and 1. • binary, for binary values, encoded as base-64 strings in profiles.

Table 26. Attributes (continued)

Attribute	Description

reboot	The default for the boolean 'reboot' attribute is "false". If a parameter change requires a reboot to take effect, then this parameter must have its 'reboot' attribute set to "true".
	AIX Runtime Expert itself never reboots systems. By default, the **artexset** command will not force the setting of reboot parameters. If the profile contains reboot parameters, the command will fail:
	`0590-502: profile has parameters that require a reboot. Profile has not been set. Use -l all flag to force set for all parameters`
	If called with the proper *–l* flag, the **artexset** command sets the value and warns the users that a reboot is required for changes to take effect:
	`0590-206 A manual post-operation is required for thechanges to take effect Please reboot the system`

priority	By default, parameters are set in no defined order by the **artexset** command. The *priority* attribute can be used alter this behavior and force a parameter to be set before or after other parameters. The default priority is 0. The priority attribute can be used to change this default priority to any integer value between -2147483648 and 2147483647. Parameters with a higher priority are executed before parameters with a lower priority. The order in which parameters with the same priority are set is undefined.
targetClass	Some parameters have to be associated with a target, as explained in section on "The Target element" of a profile. Those parameter must have their *targetClass* attribute set to the coma-separated list of their supported target classes.
cfgmethod	A **<ParameterDef>** element can inherit command line elements from a **<CfgMethod>** element by referencing this configuration method *id* attribute with the *cfgmethod* attribute. For more information on configuration methods, refer to the section on <CfgMethod> element.
constraint	A **<ParameterDef>** element can use the constraint attribute to reference the id attribute of a **<ConstraintDef>** element, indicating that the parameter is subjected to the constraint. For more information on constraints, refer to the section on "The **<ConstraintDef>** element".

Examples

1. Following is an example of a parameter definition with an alternative integer type: *kernel_heap_size*, from the `vmoParam.xml` catalog file:

   ```
   <ParameterDef name="kernel_heap_psize" type="integer-bi">
   ```

   When extracting the value of this parameter through an **artexget** command, the result is something like (excerpted from the resulting profile).

   ```
   <Parameter name="kernel_heap_psize" value="16M"… />
   ```

   The parameter value will be interpreted differently, depending on the type:

   - Since it is declared to be of type integer-bi, the value is 16M= 16,777,216.
   - If the type had been integer-si, the value would have been “16M”=16,000,000.
2. Example of a binary parameter: the trusted signature database `tsd.dat` in the `tsdParam.xml` catalog:

   ```
   <ParameterDef name="tsdatabase" type="binary">
   ```
3. Example of a parameter with a *reboot* attribute. The type of dump parameter in the `sysdumpdevParam.xml` catalog:

   ```
   <ParameterDef name="type_of_dump" type="string" reboot="true">
   ```
4. Example of a parameter with one target class: the *addr* parameter from the `mktcpipParam.xml` catalog applies to a specific network interface:

   ```
   <ParameterDef name="addr" type="string" cfgmethod="mktcpip" targetClass="interface">
   ```
5. Example of a parameter with several target classes: the naming specification parameter from the `coreParam.xml` applies to a specific user (root, admin, guest, etc.) in a specific registry (files, LDAP).

   ```
   <ParameterDef name="namingspecification" type="string" reboot="true"
   targetClass="user,registry"
   cfgmethod="coremgt">
   ```
6. Example of use of the *cfgmethod* attribute: For the **<Get type="current">** operation, the fixed parameter of the `chlicenseParam.xml` catalog inherits the **<Command>** element from the *chlicense* configuration method, but it also defines its own **<Filter>** and **<Mask>** locally for this same operation:

   ```
   <CfgMethod id="chlicense">
     <Get type="current">
       <Command>lslicense -c -A</Command>
     </Get>
   </CfgMethod>
   <ParameterDef name="fixed" cfgmethod="chlicense" type="integer">
     <Get type="current">
       <Filter>tail -n 1 | cut -d: -f3</Filter>
       <Mask value="1">(.*)</Mask>
     </Get>
   </ParameterDef>
   ```
7. Example of usage of the constraint attribute: the authorizations parameter of the `authParam.xml` catalog is subjected to the **setkst** constraint defined earlier in a **<ConstraintDef>** element:

   ```
   <ParameterDef name="authorizations" cfgmethod="cat" constraint="setkst" type="string">
   ```

name attribute
The name of a parameter is often dictated by the command used to get or set the parameter.

Parameter names must be unique within a catalog file. This is required to ensure that a **<Parameter>** element in a profile can be associated with a unique **<ParameterDef>** element in a catalog file.

- If the **get** command displays several parameter-value pairs, then the **<Mask>** element can be used to extract multiple parameters from a single command output. This is only possible if the name of the parameter matches the name used in the output of the **get** command.
- If the **set** command accepts several parameter-value pairs, then the %n and %v1 sequences can be used in an **<Argument>** element to set multiple parameters with a single command. This is only possible if the name of the parameter matches the name used by the **set** command.

Examples

1. Example: The **raso -a** command used in the `rasoParam.xml` catalog displays one parameter per line of display:

   ```
   kern_heap_noexec = 0
   kernel_noexec = 1
   mbuf_heap_noexec = 0
   mtrc_commonbufsize = 485
   ```

   In this easy case, the parameter names will be *kernel_heap_noexec*, *kernel_noexec*, etc.
2. Example: The command used in the **get** configuration method of the `acctctlParam.xml` catalog displays a result that is more difficult to parse. Not only is the name of the parameter integrated into a non-formatted sentence, but both the parameter names and their values are localized. The get configuration methods will have to run the command while setting the environment variable **LANG=C**, and, in each line, replace the key words by pertinent parameter names:

   ```
   Advanced Accounting is not running.
   Email notification is off.
   The current email address to be used is not set.
   Recover CPU accounting time in turbo mode is False.
   ```

   In the above example, the variable names that have been chosen are `accounting`, `email`, `email_adddr` and `turacct`.

Related information

- The **<Parameter>** element
- The **<Mask>** element
- Expansion of command line elements

<ConstraintDef> element

Syntax

Parent element: **<Catalog>**, **<ParameterDef>**

The following attributes are supported:

Table 27. Attributes

Attribute	Required	Type	Description
id	no*	string	Specifies the name of the parameter constraint.

*This attribute must be specified for **<Constraint>** elements defined at the catalog level.

The following child elements are supported:

Table 28. Child Elements		
Child Elements	**Required**	**Description**
<Description>	no	Textual description of the disruptive command.
<PreOp>	no	Disruptive operations to run before setting the parameter value.
<PostOp>	no	Disruptive operations to run after setting the parameter value.
<BuiltIn>	no	Built-in disruptive operation This element may occur multiple times.

Usage

Some tuning and configuration parameters may require disruptive operations for value changes to take effect. A disruptive operation is any operation that may temporarily interrupt access to a service or a device. Typical disruptive operations are restarting a daemon, unmounting or mounting a filesystem, bringing a network adapter card offline or online. The AIX Runtime Expert program uses constraints to show that a parameter requires a disruptive operations for changes to take effect. A **<ConstraintDef>** element is used to define such a constraint.

The constraint can be defined either:

- Inside a **<ParameterDef>** element, if the constraint only applies to the single parameter.
- At the catalog level, the **<ConstraintDef>** element must have an *id* attribute to allow the constraint to be referenced later in **<ParameterDef>** elements.

Built-In constraint

The **<BuiltIn>** element does not contain any attribute or child element.

The built-in constraint defines operations that are hard coded in the core engine. There is currently only one built-in constraint defined: *bosboot*. The difference of built-in constraints with other disruptive operations is that the *bosboot* command is never run by AIX Runtime Expert. The core engine will only warn that a *bosboot* is required for changes to take effect.

```
0590-206 A manual post-operation is required for the changes to take effect
Please perform a bosboot
```

PreOp and PostOp constraint

The **<PreOp>** element defines mandatory commands (shell expressions) to be run before the parameter value is set by the set configuration method. The **<PostOp>** element defines mandatory commands to be run after the execution of the set configuration method.

An **<ConstraintDef>** element must contain 0 or one **<PreOp>** child element, and 0 or one **<PostOp>** child element.

Examples

1. Example of a built-in constraint (at catalog level)

```
<ConstraintDef id="bosboot">
  <Description>
<NLSCatalog catalog="artexcat.cat" setNum="51" msgNum="3">
    bosboot
```

```
</NLSCatalog>
  </Description>
  <Built>Inbosboot</BuiltIn>
</ConstraintDef>
```

2. Example of **<PreOp>** constraint: The *clic* constraint in the `trustchkParam.xml` catalog. Note that in this example, the **preop** command does not run anything, but only checks the presence of a kernel extension required by the **set** command. If the kernel extension is not installed, then the constraint defined in the **<PreOp>** element will fail and the **set** command will not be run:

```
<ConstraintDef id="clic">
  <Description>
  <NLSCatalog catalog="artexcat.cat" setNum="48" msgNum="3">
     Check that the clic.rte kernel extension is installed.
  </NLSCatalog>
  </Description>
  <Pre>Oplslpp -l "clic*"</PreOp>
 </ConstraintDef>
```

3. Example of **<PostOp>** constraint: The set Kernel Security Tables constraint in the `authParam.xml` catalog. The modified databases need to be loaded only once in the kernel after all modifications have been made.

```
<ConstraintDef id="setkst">
  <Description>
<NLSCatalog catalog="artexcat.cat" setNum="5" msgNum="3">
Send the authorizations database to the KST (Kernel Security Tables)
    </NLSCatalog></Description>
  <PostOp>/usr/sbin/setkst -t auth &gt;/dev/null</PostOp>
  </ConstraintDef>
```

<CfgMethod> element

Syntax

Parent element: **<Catalog>**

The following attribute is supported:

Table 29. Attribute

Attribute	Required	Type	Description
id	yes	string	Specifies the name of the configuration method.

The following child elements are supported:

Table 30. Child elements

Child elements	Required	Number	Description
<Get>	no	0 – 1	Configuration method definition for the get operation. This element may occur multiple times.
<Set>	no	0 – 1	Configuration method definition for the set operation. This element may occur multiple times.

Table 30. Child elements (continued)			
Child elements	**Required**	**Number**	**Description**
<Diff>	no	0 – 1	Configuration method definition for **diff** operation.
<Discover>	no	0 – 1	Configuration method definition for target discovery.
<Property>	no	0 – any	Assigns a property to the parameters using the configuration method.

Usage

The **<CfgMethod>** element defines a configuration method that can later be referenced by a parameter using the *cfgmethod* attribute of the **<ParameterDef>** element. The parameter then inherits all the elements defined under the referenced configuration method.

Depending on the parameter, using a configuration may offer several advantages over the local definition:

- It simplifies the catalog file, avoiding duplication of the same command line elements for several parameters.
- It allows multiple parameters to be treated by a single command.

Example

The `vmoParam.xml` catalog defines a lot of parameters that all use the same configuration method. Here is a simplified version of this catalog:

```
<Catalog id="vmoParam" version="2.1">
 <CfgMethod id="vmo">
    <Get type="current">
      <Command>/usr/sbin/vmo -a</Command>
      <Mask name="1" value="2">[[:space:]]*(.*) = (.*)</Mask>
    </Get>

    <Get type="nextboot">
      <Command>/usr/sbin/vmo -r -a</Command>
      <Mask name="1" value="2">[[:space:]]*(.*) = (.*)</Mask>
    </Get>

    <Set type="permanent">
      <Command>/usr/sbin/vmo -p%a</Command>
   <Argument>%n=%v1</Argument>
    </Set>

    <Set type="nextboot">
      <Command>/usr/sbin/vmo -r%a</Command>
      <Argument>%n=%v1</Argument>
    </Set>
  </CfgMethod>

  <ParameterDef name="ame_maxfree_mem" cfgmethod="vmo" type="integer" />
  <ParameterDef name="ame_min_ucpool_size" cfgmethod="vmo" type="integer" />
  <ParameterDef name="ame_minfree_mem" cfgmethod="vmo" type="integer" />
  ...

</Catalog>
```

Related Information

- Command line generation
- The **<Get>** element

- The **<Set>** element

<Get> element

Syntax

Parent element: **<CfgMethod>**, **<ParameterDef>**

The following attribute is supported:

Table 31. Attribute			
Attribute	**Required**	**Type**	**Description**
type	yes	string	Specifies the type of the **get** command (current or *nextboot*).

The following child elements are supported:

Table 32. Child elements			
Child elements	**Required**	**Number**	**Description**
<Command>	no	0 – 1	Command
<Argument>	no	0 – 1	Command-line arguments
<Stdin>	no	0 – 1	Arguments supported by the **<Stdin>** element
<Filter>	no	0 – 1	Filter
<Mask>	no	0 – 1	Output capturing mask
<Prereq>	no	0 – any	Assigns a prerequisite to the get operation

The **<Command>** element must be defined for each parameter, either at the **<CfgMethod>** level or directly at the **<ParameterDef>** level.

Usage

The **<Get>** element describes how the value of a particular parameter is captured. It can be used either directly under the **<ParameterDef>** element, or under a **<CfgMethod>** element referenced in the **<ParameterDef>** element using the *cfgmethod* attribute, or using a combination of those two possibilities.

Two Get elements should be defined for each parameter, one for each supported value of the *type* attribute:

- Get **type="current"** identifies the method that will be run to retrieve the runtime value of the parameter.
- Get **type="nextboot"** identifies the method that will be run to retrieve the value that the parameter will have after the next reboot of the system.
- The get method to be run depends on the operation being performed:
  - If the **artexget** command is called with the *–r* flag, the current get method is used.
  - If the **artexget** command is called with the *–n* flag, the *nextboot* get method is used.
  - If the **artexget** command is called with the *–p* flag, the method run depends on the parameters input to the *applyType* attribute. The current get method is used for the parameters that have their *applyType* attribute set to runtime and the *nextboot* get method is used for the parameters that have an *applyType* attribute of reboot.

Related Information

Command line generation

The **<Mask>** element.

<Set> element
The **<Set>** element defines how to build a command line to set the value of a parameter.

Syntax

Parent element: **<CfgMethod>**, **<ParameterDef>**

The following attribute is supported:

Table 33. Attribute

Attribute	Required	Type	Description
type	yes	string	Specifies the type of the **set** command as current or nextboot.

The following child elements are supported:

Table 34. Child elements

Child elements	Required	Number	Description
<Command>	no	0 – 1	Command
<Argument>	no	0 – 1	Command-line arguments
<Stdin>	no	0 – 1	Stdin arguments
<Prereq>	no	0 – any	Assigns a prerequisite to the **<Set>** operation

Note: The **<Command>** element must be defined for each parameter, either at the **<CfgMethod>** level or directly at the **<ParameterDef>** level.

Usage

There are three types of **<Set>** elements that can be defined for each parameter, identified by their required *type* attribute:

- Set **type="current"** defines a set operation that only changes the value of the parameter for the current session. Any change made using the set operation will be lost after a reboot of the system.
- Set **type="nextboot"** defines a set operation that only changes the value the parameter will take after the next reboot of the system. The current value is not modified.
- Set **type="permanent"** defines a set operation that changes both the current and the nextboot value of the parameter.

The type of the set operation run is decided based on parameters included when the **artexset** command is run, based the parameter *applyType* attribute in the profile. The following table summarizes the set methods that are run, depending on the set methods defined in the catalog file depending on the *applyType* attribute for the parameter:

Table 35. Set Methods - set method types defined and Parameter applyType attribute.

current	nextboot	permanent	runtime	nextboot
0	0	0	not set (error)	not set (error)

Table 35. Set Methods - set method types defined and Parameter applyType attribute. (continued)				
current	**nextboot**	**permanent**	**runtime**	**nextboot**
0	0	1	set permanent	not set (error)
0	1	0	set nextboot + warning	set nextboot
0	1	1	set permanent	not set (error)
1	0	0	set current + warning	set nextboot
1	0	1	set permanent	not set (error)
1	1	0	set current set nextboot	set nextboot
1	1	1	set permanent	set nextboot

Related Information

Command line generation.

<Diff> element
The **<Diff>** element defines how to build a command line to compare two values of a parameter.

Syntax

Parent element: **<CfgMethod>**, **<ParameterDef>**

The following child elements are supported:

Table 36. Child elements		
Child elements	**Required**	**Description**
<Command>	no	Command
<Argument>	no	Command-line arguments
<Stdin>	no	Stdin arguments
<Filter>	no	Filter
<Mask>	no	Output capturing mask

Note: The **<Command>** element must be defined for each parameter, either at the **<CfgMethod>** level or directly at the **<ParameterDef>** level.

Usage

The **<Diff>** element is usually not required, since the framework knows how to compare two parameter values internally based on the type (string, integer, integer-bi, binary, etc.). However, in case the internal comparison is not adapted for a particular parameter, it is possible to use an external command instead.

Example

The following **<Diff>** element can be used for most parameters, even though using the internal comparison function is more efficient. The **<Diff>** element uses the **diff** command to compare two files that contains the two values:

```
<Diff>
  <Command>/usr/bin/diff %f1 %f2; echo $?</Command>
</Diff>
```

Related Information

Command line generation.

The **<Mask>** element.

<Discover> element
The **<Discover>** element defines how to build a command line to discover targets for a parameter that supports them.

Syntax

Parent element: **<CfgMethod>**, **<ParameterDef>**

The following child elements are supported:

Table 37. Child elements

Child elements	Required	Number	Description
<Command>	no	0 – 1	Command
<Prereq>	no	0 – any	Assigns a prerequisite to the discover operation

Note: The **<Command>** element must be defined for each parameter, either at the **<CfgMethod>** level or directly at the **<ParameterDef>** level.

Usage

A discover command is used to obtain the list of target instances for a given parameter.

The output of a discover command for a parameter that supports N target classes have the following format:

```
class_1=inst_1_1;class_2=inst_2_1;...;class_N=inst_N_1
class_1=inst_1_2;class_2=inst_2_2;...;
class_N=inst_N_2class_1=inst_1_3;
class_2=inst_2_3;...;class_N=inst_N_3
...
```

The **artexget** command generates and runs a discover command for parameters that satisfy one of the following criteria:

- Contain a **<Target>** element with empty *class* and *instance* attributes. **<Target class="" instance="" />**
- Contain at least one **<Target>** element with a *match* attribute: **<Target class="..." match="..." />**

The **artexset** command additionally requires the following two criteria to be satisfied:

- The **artexset** command is called with the *–d* flag.
- The **<Parameter>** element in the profile has the *setDiscover* attribute set to true.

Examples

1. The `mktcpipParam.xml` catalog uses the following discover command to obtain the list of the network interfaces defined on the system:

```
<Discover>
  <Command>
   /usr/sbin/lsdev -C -c if -F "name" | /usr/bin/sed -e 's/^/interface=/'
 </Command>
</Discover>
```

This command gives the following output:

```
interface=en0
interface=et0
interface=lo0
```

2. The `chuserParam.xml` catalog uses the following **`discover`** command to get to the list of all users for all loadable authentication modules:

```
<Discover>
  <Command>
  /usr/sbin/lsuser -a registry ALL |
/usr/bin/sed -e "s/\(.*\) registry=\(.*\)/module=\2;user=\1/g"
</Command>
</Discover>
```

This command gives the following output:

```
module=LDAP;user=daemon
module=LDAP;user=bin
module=LDAP;user=sys
module=LDAP;user=adm
...
module=files;user=root
module=files;user=daemon
module=files;user=bin
module=files;user=sys
module=files;user=adm
...
```

<Command> element
The **<Command>** element defines the base command used to perform the operation defined by the parent element.

Syntax

Parent element: **<Get>**, **<Set>**, **<Diff>**, **<Discover>**, **<PrereqDef>**, **<Prereq>**, **<PropertyDef>**, **<Property>**, **<Command>**

Usage

The content of the **<Command>** element is expanded as described in section Expansion of command line elements and combined with the other command line elements to form a complete command line. See section Command line generation for more details.

Some characters often found in shell expressions, such as <, > and & are not allowed in XML documents. These characters must be replaced by the corresponding XML entity:

Table 38. XML entities

Character	XML entity
<	&lt;
>	&gt;
&	&amp;

Alternatively, a CDATA section can be used if the expression contains many of such characters. The CDATA sections start with **<![CDATA[** and ends with **]]>**.

The **<Command>** element must be defined for each supported operation of each parameter, either at the **<CfgMethod>** level or at the **<ParameterDef>** level.

Example

The `envParam.xml` catalog defines a parameter called profile that represents the contents of the `/etc/profile` file. For this parameter, the **<Get>** element uses the **cat** command to capture the content of the `/etc/profile` file:

```
<ParameterDef name="profile">
  <Get type="current">
    <Command>/usr/bin/cat /etc/environment</Command>
  </Get>
</ParameterDef>
```

<Argument> element

Syntax

Parent element: **<Get>**, **<Set>**, **<Diff>**, **<PrereqDef>**, **<Prereq>**, **<PropertyDef>**, **<Property>**

Usage

The content of the **<Argument>** element is expanded as described in section Expansion of command line elements and combined with the **<Command>** and or the <Stdin> elements to form a complete command line. See section Command line generation for more details.

Some characters often found in shell expressions, such as <, > and & are not allowed in XML documents. These characters must be replaced by the corresponding XML entity:

Table 39. XML entities

Character	XML entity
<	&lt;
>	&gt;
&	&amp;

Alternatively, a CDATA section can be used if the expression contains many of such characters. CDATA sections start with **<![CDATA[** and ends with **]]>**.

Example

The `vmoParam.xml` catalog uses the **<Argument>** element to add an argument to the **vmo** command for each **vmo** parameter in the profile:

```
<CfgMethod id="vmo">
  <Set type="permanent">
    <Command>/usr/sbin/vmo -p%a</Command>
    <Argument> -o %n=%v1</Argument>
  </Set>
</CfgMethod>
```

<Stdin> element

Syntax

Parent element: **<Get>**, **<Set>**, **<Diff>**, **<PrereqDef>**, **<Prereq>**, **<PropertyDef>**, **<Property>**

Usage

The content of the **<Stdin>** element is expanded as described in section Expansion of command line elements and the resulting data is written to the standard input of the command line generated for the operation defined in the parent element.

Example

The `envParam.xml` catalog defines a parameter called profile that represents the content of the `/etc/profile` file. For this parameter, the set operation writes the value of the parameter to the standard input of the **cat** command to overwrite the `/etc/profile` file:

```
<ParameterDef name="profile">
  <Set type="permanent">
    <Command>/usr/bin/cat &gt; /etc/profile</Command>
     <Stdin>%v1</Stdin>
   </Set>
 </Get>
```

Related information

Command line generation

<Filter> element

Syntax

Parent element: **<Get>**, **<Diff>**, **<PropertyDef>**, **<Property>**

Usage

The content of the **<Filter>** element is a command to which the output of the command line generated for the operation defined in the parent element is passed as input.

Some characters often found in shell expressions, such as <, > and & are not allowed in XML documents. These characters will need to be replaced by the corresponding XML entity:

Table 40. XML entities

Character	XML entity
<	&lt;
>	&gt;
&	&amp;

Alternatively, a CDATA section can be used if the expression contains many of such characters. The CDATA sections start with **<![CDATA[** and ends with **]]>**.

Example

The `nfsParam.xml` catalog uses the **<Filter>** element for the get operation of parameter *v4_root_node* to extract the root node from the output of the **nfds -getnode** command:

```
<ParameterDef id="v4_root_node">
  <Get type="current">
    <Command>
       /usr/sbin/nfsd -getnodes
    </Command>
    <Filter>
       /usr/bin/awk -F: 'NR == 2 { printf("%s", $1) }'
    </Filter>
  </Get>
</ParameterDef>
```

Related information

Command line generation

Syntax

Parent element: **<Get>**, **<Diff>**, **<Discover>** (only under a <**SeedDef**>),**<PropertyDef>**, **<Property>**

The following attributes are supported when used in a **<Get>** or **<Diff>** element:

Table 41. Attributes

Attribute	Required	Type	Description
name	no	integer	Specifies the index of the subexpression that matches the name of the parameter. Valid values are 1 and 2.
value	no	integer	Specifies the index of the subexpression that matches the value of the parameter. Valid values are 1 and 2.

The following attributes are supported when used under the **<Discover>** subelement of a **<SeedDef>** element:

Table 42. Attributes

Attribute	Required	Type	Description
catalog	yes	integer	Specifies the index of the subexpression that matches the name of the catalog. Valid values are 1, 2, and 3.
name	yes	integer	Specifies the index of the subexpression that matches the name of the parameter. Valid values are 1, 2, and 3.
target	no	integer	Specifies the index of the subexpression that matches the target of the parameter. Valid values are 1, 2, and 3.

The following attribute is supported when used under the **<PropertyDef>** or **<Property>** element:

Table 43. Attribute

Attribute	Required	Type	Description
value	no	integer	Specifies the index of the subexpression that matches the name of the parameter. Must be set to “1” if specified.

Usage

The **<Mask>** element defines a regular expression that is applied on each line of command output to extract data from those lines. The data that is extracted depends on where the **<Mask>** element is used.

If no attributes are specified, the last line in the command output that matches the regular expression is used to extract the data. The extracted data is the part of the line that matches the regular expression. When used under a **<Get>** or **<Diff>** element, the extracted data is used as the parameter value. When used under a **<PropertyDef>** or **<Property>** element, the extracted data is used as the property value.

If only the *value* attribute is specified, it must be set to 1, and the regular expression must contain only one subexpression. The last line in the command output that matches the regular expression is used to extract the data. The extracted data is the part of the line that matches the first (and only) subexpression. When used under a **<Get>** or **<Diff>** element, the extracted data is used as the parameter value. When used under a **<PropertyDef>** or **<Property>** element, the extracted data is used as the property value.

If the *name* and *value* attributes are specified, one of those attributes must be set to 1 and the other must be set to 2, and the regular expression must contain two subexpressions. A *name* and a *value* are extracted from each line of the command output that matches the regular expression. When used in a **<Get>** element, the name is used as the parameter name and the value as the parameter value. When used in a **<Diff>**element, the name is used as the parameter name and the value is used as the comparison result. Using this function, the values of several parameters can be extracted by using a single **get** command, and several parameters can be compared by using a single **diff** command.

When used in the **<Discover>** subelement of a **<SeedDef>** element, the catalog and name attributes must be specified. A catalog name and a parameter name are extracted from each line of the command output that matches the regular expression. If a catalog that matches the extracted catalog name is found on the system, and if it contains a definition for a parameter that matches the extracted parameter name, a parameter is inserted into the profile. The optional target argument can be added to extract a target definition for each discovered parameter. The target definition must follow the semicolon-separated list of `class=instance` pairs format, such as `class1=instance1;class2=instance2;...` format.

Examples

1. The `vmoParam.xml` catalog uses the **<Mask>** element with the *name* and *value* attributes to extract all parameter values from a single **vmo -a** command:

```
<CfgMethod id="vmo">
  <Get type="current">
    <Command>/usr/sbin/vmo -a</Command>
    <Mask name="1" value="2">[[:space:]]*(.*) = (.*)</Mask>
  </Get>
</CfgMethod>
```

2. Had the `vmoParam.xml` catalog been written in a way that one separate command was used to capture the value of each parameter, then the **<Mask>** element could have been used with just the *value* attribute set and no *name* attribute:

```
<CfgMethod id="vmo">
  <Get type="current">
    <Command>/usr/sbin/vmo -o %n</Command>
    <Mask value="1"> = (.*)</Mask>
  </Get>
</CfgMethod>
```

3. Or, by using a regular expression that matches only the value:

```
<CfgMethod id="vmo">
  <Get type="current">
    <Command>/usr/sbin/vmo -o %n</Command>
    <Mask>[^ ]*$</Mask>
  </Get>
</CfgMethod>
```

From the three examples above, the first is the most efficient, since it requires only a single command to capture all the **vmo** command parameters. Examples 2 and 3 would generate a separate command for each **vmo** command parameter, since the parameter name is used in the **<Command>** element.

4. The following **<SeedDef>** element defines a seed that can be used to discover all attributes of all devices. It uses a target to designate the device they operate on:

```
<SeedDef name="devAttr">
  <Discover>
    <Command>
      /usr/sbin/lsdev -F 'name class subclass type' |
      while read DEV CLASS SUBCLASS TYPE
      do
        /usr/sbin/lsattr -F attribute -l $DEV |
        while read PAR
        do
          echo device=$DEV devParam.$CLASS.$SUBCLASS.$TYPE $PAR
        done
      done
    </Command>
    <Mask target="1" catalog="2" name="3">(.*) (.*) (.*) <Mark>
  </Discover>
</SeedDef>
```

The discovery command prints each discovered device attribute on a separate line, by using the following format:

```
device=DeviceName devParam.Class.Subclass.Type AttributeName
```

For example,

```
device=en0 devParam.if.EN.en tcp_recvspace
device=en0 devParam.if.EN.en tcp_sendspace
device=ent0 devParam.adapter.vdevice.IBM,l-lan alt_addr
device=ent0 devParam.adapter.vdevice.IBM,l-lan chksum_offload
```

Related information

Command line generation

<SeedDef> element
The **<SeedDef>** element defines a seed that can be used in a profile by using a **<Seed>** element.

Syntax

Parent element: **<Catalog>**

The following attribute is supported:

Table 44. Attribute

Attribute	Required	Type	Description
name	yes	string	Specifies the name of the seed. This name must be unique for each catalog.

The following child element is supported:

Table 45. Child element

Child element	Required	Description
<Discover>	yes	Specifies the command that is used to discover parameters.

Usage

Seeds are used to discover parameters dynamically during a get operation.

When the **artexget** command is issued, each **<Seed>** element in the input profile is expanded into one or more **<Parameter>** elements, based on the rules defined in the matching **<SeedDef>** element of the catalog file. This process is called parameter discovery. The **artexget** command then proceeds as usual with the expanded profile.

The **<SeedDef>** element contains only a **<Discover>** subelement, that defines a command to run, and a mask to extract parameter names, catalog names (as colon-separated lists, without the **.xml** extension), and optionally targets from the output of the command (by using the *class1=instance1;class2=instance2;...* format). For each line of the output, the first catalog from the colon-separated list that is found on the system is loaded. If a parameter definition is found in this catalog, then a parameter is created in the output profile that has the targets that were extracted from the line. Lines from the command output that do not match the mask, or for which no catalog file is found, or that have no parameter definition if found in the catalog file, are ignored.

Examples

1. The following catalog defines a **<SeedDef>** element called *vmoTunables* that discovers all the nonrestricted *vmo tunables* seed supported by AIX Runtime Expert:

```
<?xml version="1.0" encoding="UTF-8"?>
<Catalog id="vmoSeed">
  <SeedDef name="vmoTunables">
    <Discover>
      <Command>/usr/sbin/vmo -x | /usr/bin/awk -F, '{ print "vmoParam:" $1 }'</Command>
      <Mask catalog="1" name="2">(.*):(.*)/Mask>
    </Discover>
  </SeedDef>
</Catalog>
```

   The discovery command prints each tunable on a separate line, preceded by the name of the catalog that defines the tunables:

```
...
vmoParam:enhanced_affinity_vmpool_limit
vmoParam:esid_allocator
vmoParam:force_relalias_lite
vmoParam:kernel_heap_psize
...
```

   The following profile uses the *vmo tunables* seed to capture all the nonrestricted *vmo tunables* seed supported by AIX Runtime Expert:

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile>
  <Catalog id="vmoSeed">
    <Seed name="vmoTunables"/>
  </Catalog>
</Profile>
```

   When the **artexget -r** command is run on the profile, the command generates a profile similar to the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile>
 <Catalog id="vmoParam">
  ...
  <Parameter name="enhanced_affinity_vmpool_limit" value="10"/>
  <Parameter name="esid_allocator" value="0"/>
  <Parameter name="force_relalias_lite" value="0"/>
  <Parameter name="kernel_heap_psize" value="65536" applyType="nextboot" reboot="true"/>
  ...
</Catalog>
</Profile>
```

2. The following **<SeedDef>** element defines a seed that is used to discover all attributes of all devices. The element uses a target seed to designate the device they operate on:

```
<SeedDef name="devAttr">
  <Discover>
    <Command>
      /usr/sbin/lsdev -F 'name class subclass type' |
      while read DEV CLASS SUBCLASS TYPE
      do
        /usr/sbin/lsattr -F attribute -l $DEV |
        while read PAR
        do
          echo device=$DEV devParam.$CLASS.$SUBCLASS.$TYPE:devParam.$CLASS
.$SUBCLASS:devParam.$CLASS $PAR
        done
      done
    </Command>
    <Mask target="1" catalog="2" name="3">(.*) (.*) (.*)</Mask>
  </Discover>
</SeedDef>
```

The discovery command prints each discovered device attribute on a separate line, by using the following format:

```
device=DeviceName devParam.Class.Subclass.Type:devParam.Class.Subclass:devParam.Class
AttributeName
```

For example:

```
device=en0 devParam.if.EN.en:devParam.if.EN:devParam.if tcp_recvspace
device=en0 devParam.if.EN.en:devParam.if.EN:devParam.if tcp_sendspace
device=ent0 devParam.adapter.vdevice.IBM,l-lan:devParam.adapter.vdevice:devParam.adapter
alt_addr
device=ent0 devParam.adapter.vdevice.IBM,l-lan:devParam.adapter.vdevice:devParam.adapter
chksum_offload
```

<Prereq> element
The **<Prereq>** element assigns a prerequisite to **<Get>**, **<Set>**, and **<Discover>** operations.

Syntax

Parent element: **<Get>**, **<Set>**, and **<Discover>**

The following attribute is supported:

Table 46. Attribute

Attribute	Required	Type	Description
id	no	string	Specifies an unique identifier

The following child elements are supported:

Table 47. Child elements

Child element	Required	Description
<Command>	no	Command
<Argument>	no	Command-line arguments
<Stdin>	no	Arguments supported by the <Stdin> element
<ErrMessage>	no	Message to print if prerequisites fail

Note: The **<Command>** element must be defined for each prerequisite: at the **<ParameterDef>** level, at the **<CfgMethod>** level, or in a **<PrereqDef>** element.

Usage

Prereqs are commands that condition the processing of a **<Get>**, **<Set>**, and **<Discover>** operation for parameters that use this **<Get>**, **<Set>**, and **<Discover>** operations. Parameters for which a **prereq** command fails (nonzero return code) are ignored, and the error message defined in the prerequisite is displayed.

The **<Prereq>** element assigns a prerequisite to the parent **<Get>**, **<Set>**, or **<Discover>** operation. The prerequisite is either defined locally under the **<Prereq>** element, or inherited from a higer-level **<Prereq>** or **<PrereqDef>** element that has a matching *id* attribute.

A parameter has all the prerequisites defined locally under the **<ParameterDef>** element. Also, the prerequisite has the properties defined in the configuration method of the parameter, if a configuration method is used. The consequence is that if a prerequisite is defined in a **<CfgMethod>** element, all the **<ParameterDef>** elements that use the configuration method will automatically have that prerequisite (although some of those elements might redefine the prerequisite locally).

The **<Command>**, **<Argument>**, **<Stdin>**, and **<ErrMessage>** elements that define a prerequisite for a given operation are searched in this order:

- Under the **<Prereq>** subelement of the relevant operation of the **<ParameterDef>** element.
- If the **<ParameterDef>** element has a *cfgmethod* attribute, under the **<Prereq>** subelement that has a matching *id* of the relevant operation of the configuration method.
- Under the **<PrereqDef>** element of the catalog that has a matching id.

Example

The following example defines a prerequisite that checks that the **netaddr** and **netaddr6** parameters are applied on the same system on which they were captured:

```
<ParameterDef name="netaddr" type="string" targetClass="device" cfgmethod="attr">
  <Set type="permanent">
    <Prereq>
      <Command>[[ `/usr/bin/uname -f` = %p[nodeId] ]]</Command>
      <ErrMessage>Parameter cannot be applied to a different node</ErrMessage>
    </Prereq>
  </Set>
</ParameterDef>

<ParameterDef name="netaddr6" type="string" targetClass="device" cfgmethod="attr">
  <Set type="permanent">
    <Prereq>
      <Command>[[ `/usr/bin/uname -f` = %p[nodeId] ]]</Command>
      <ErrMessage>Parameter cannot be applied to a different node</ErrMessage>
    </Prereq>
  </Set>
</ParameterDef>
```

In this example, the test is run twice: once for the **netaddr** parameter, and once for the **netaddr6** parameter. This dual processing is because each parameter has its own prerequisite with its own **<Command>** element. See , artex_catalog_elem_PrereqDef.dita for an example that requires only one run of the test.

Related Information

- artex_catalog_cmdline_gen.dita
- artex_catalog_elem_PrereqDef.dita

<PrereqDef> element
The **<PrereqDef>** element that can later be used in a **<Prereq>** element.

Syntax

Parent element: **<Catalog>**

The following attribute is supported:

Table 48. Attribute

Attribute	Required	Type	Description
name	yes	string	Specifies the name of the property.

The following child elements are supported:

Table 49. Child elements

Child element	Required	Description
<Command>	no	Command
<Argument>	no	Command-line arguments
<Stdin>	no	Arguments that are supported by the <Stdin> element
<ErrMessage>	no	Message to print if prerequisite fails

Note: The **<Command>** element must be defined for each prerequisite: at the **<ParameterDef>** level, at the **<CfgMethod>** level, or in a **<PrereqDef>** element.

Usage

Prereq are commands that condition the run of the **<Get>**, **<Set>**, and **<Discover>** operations for parameters that use the **<Get>**, **<Set>**, or **<Discover>** operation. Parameters for which a **prereq** command fails (nonzero return code) are ignored, and the error message defined in the prerequisite is displayed.

The **<PrereqDef>** element defines a prerequisite. This prerequisites can later be associated with an operation of a parameter or a configuration method by using a **<Prereq>** element that has the same *id* attribute.

Example

The following example defines the *nodeId* prerequisite and assigns it to the **netaddr** and **netaddr6** parameters:

```
<PrereqDef id="nodeId">
  <Command>[[ `/usr/bin/uname -f` = %p[nodeId] ]]</Command>
  <ErrMessage>Parameter cannot be applied to a different node</ErrMessage>
</PrereqDef>

<ParameterDef name="netaddr" type="string" targetClass="device" cfgmethod="attr">
  <Set type="permanent">
    <Prereq id="nodeId"/>
  </Set>
  <Property name="nodeId"/>
</ParameterDef>

<ParameterDef name="netaddr6" type="string" targetClass="device" cfgmethod="attr">
  <Set type="permanent">
    <Prereq id="nodeId"/>
  </Set>
  <Property name="nodeId"/>
</ParameterDef>
```

In this example, the test is executed only once, because the two parameters use the same **<Command>** element for their prerequisites, and the generated command line is the same for the two parameters.

Related Information

- artex_catalog_cmdline_gen.dita
- artex_catalog_elem_Prereq.dita

<Property> element
The **<Property >** element assigns a property to a parameter or a configuration method.

Syntax

Parent element: **<CfgMethod>**, **<ParameterDef>**

The following attribute is supported:

Table 50. Attribute

Attribute	Required	Type	Description
name	yes	string	Specifies the name of the property.

The following child elements are supported:

Table 51. Child element

Child element	Required	Description
<Command>	no	Command
<Argument>	no	Command-line arguments
<Stdin>	no	Arguments supported by the <Stdin> element
<Filter>	no	Filter
<Mask>	no	Output capturing mask

Note: The **<Command>** element must be defined for each property: at the **<ParameterDef>** level, at the **<CfgMethod>** level, or in a **<PropertyDef>** element.

Usage

Properties are key-value pairs that are associated with a parameter. The value of the key-value pairs is retrieved by the **artexget -r** and **artexget -n** commands and saved in the output profile. Property values saved in a profile can be inserted into a command line by using the %p[property_name] sequence.

The **<Property>** element assigns a property to a parameter or to a configuration method. The property is either defined locally under the **<Property>** element, or inherited from a higher-level **<Property>** or **<PropertyDef>** element that has a matching name attribute.

A parameter has all the properties defined locally under the **<ParameterDef>** element. Also, the parameter has all the properties defined under the parameters configuration method, if a configuration method is used. The consequence is that if a property is defined under a **<CfgMethod>** element, all the **<ParameterDef>** elements that use the configuration method will automatically have that property (although some of them might redefine the property locally).

Property values are extracted from the output of a command line. The command line is built by combining the **<Command>**, **<Argument>**, **<Stdin>**, and **<Filter>** elements as described in the Command line generation section. You must use one of the following property values: the raw output of the command line or the portion of the output that matches the mask, if a **<Mask>** element is specified.

The **<Command>**, **<Argument>**, **<Stdin>**, **<Filter>**, and **<Mask>** elements that define a property are searched in this order:

- Under the **<Property>** element at the **<ParameterDef>** level.
- If the **<ParameterDef>** element has a *cfgmethod* attribute, under the configuration method of the **<Property>** element that has a matching *name* attribute.
- Under the **<PropertyDef>** element of the catalog that has a matching name attribute.

Example

The following example assigns a *nodeId* property to the **netaddr** and **netaddr6** parameters:

```
<ParameterDef name="netaddr" type="string" targetClass="device" cfgmethod="attr">
  <Property name="nodeId">
    <Command>/usr/bin/uname -f/<Command>
    <Mask>.*/<Mask>
  </Property>
</ParameterDef>

<ParameterDef name="netaddr6" type="string" targetClass="device" cfgmethod="attr">
  <Property name="nodeId">
    <Command>/usr/bin/uname -f</Command>
    <Mask>.*</Mask>
  </Property>
</ParameterDef>
```

In this example, the mask matches the whole line and is only used to exclude the *newline* character at the end of the command output.

In this example, the **uname** command is run twice: once for the **netaddr** parameter, and once for the **netaddr6** parameter. The command is run twice because each parameter has its own property with its own **<Command>** element. See artex_catalog_elem_PropertyDef.dita, for an example that requires only one run of the **uname** command.

Related Information

- artex_catalog_cmdline_gen.dita
- artex_catalog_cmdline_gen_expansion.dita
- artex_catalog_elem_PropertyDef.dita

<PropertyDef> element
The **<PropertyDef>** element defines a property that can be used in a **<Property>** element.

Syntax

Parent element: **<Catalog>**

The following attribute is supported:

Table 52. Attribute

Attribute	Required	Type	Description
name	yes	string	Specifies the name of the property.

The following child elements are supported:

Table 53. Child element

Child element	Required	Description
<Command>	no	Command
<Argument>	no	Command-line arguments
<Stdin>	no	Arguments supported by the <Stdin> elements

Table 53. Child element (continued)		
Child element	**Required**	**Description**
<Filter>	no	Filter
<Mask>	no	Output capturing mask

Note: The **<Command>** element must be defined for each property: at the **<ParameterDef>** level, at the **<CfgMethod>** level, or in a **<PropertyDef>** element.

Usage

Properties are key-value pairs associated with a parameter. The value of the key-value pairs are retrieved by the **artexget -r** and **artexget -n** commands and saved in the output profile. Property values saved in a profile can be inserted into a command line by using the %p[property_name] sequence.

The **<PropertyDef>** element defines a property. This property can later be associated to a parameter or configuration method by using a **<Property>** element that has the same name attribute.

Example

The following example assigns a *nodeId* property to the **netaddr** and **netaddr6** parameters:

```
<PropertyDef name="nodeId">
  <Command>/usr/bin/uname -f</Command>
  <Mask>.*</Mask>
</PropertyDef>

<ParameterDef name="netaddr" type="string" targetClass="device" cfgmethod="attr">
  <Property name="nodeId"/>
</ParameterDef>

<ParameterDef name="netaddr6" type="string" targetClass="device" cfgmethod="attr">
  <Property name="nodeId"/>
</ParameterDef>
```

In this example, the **uname** command is run only once, because the two parameters use the same **<Command>** element for their property, and the generated command line is the same for the two parameters.

Related Information

- artex_catalog_cmdline_gen.dita
- artex_catalog_cmdline_gen_expansion.dita
- artex_catalog_elem_Property.dita

Command-line generation

The AIX Runtime Expert framework relies on external commands to capture, set and optionally compare parameter values. This topic explains how command lines are built based on the syntax information provided in the catalog files.

Operations

For each parameter, the following operations can be defined:

- Get **type="current"**, used to capture the current value of the parameter.
- Get **type="nextboot"**, used to capture the value the parameter that the parameter will have after a reboot.
- Set **type="current"**, used to set the current value of the parameter. This parameter value is lost upon reboot.

- Set **type="nextboot"**, used to set the value the parameter that the parameter will have after a reboot.
- Set **type="permanent"**, used to set the current value of the parameter, knowing that this value will persist after a reboot.
- **diff** operation, used to compare two values of the parameter.
- Discover operation, used to find targets for parameters that support them.
- Property, used to capture a property for a parameter.
- Prerequisite, used to condition the execution of a get, set, or discover operation for a given parameter.

Not all operations need to be defined for all parameters. The two **get** operations and all the **set** operations supported by the parameters must be defined. The **diff** operation is optional, and if it is not defined, comparisons between parameter values are done internally based on the parameter type, such as string, and integer. The **discover** operation must be defined only for parameters that have targets. Properties and prerequisites are only defined when needed.

Command line elements

For each operation supported by a parameter, up to five different elements can be used to define how a command line can be built to perform the operation:

- **<Command>** element, used to define the base command, for handling parameters.
- **<Stdin>** element, used to define the data that will be written to the command line standard input.
- **<Argument>** element, used to insert parameter specific data into a **<Command>** or **<Stdin>** element.
- **<Filter>** element, used to filter the output a command line for the **get** and **diff** operations.
- **<Mask>** element, used to extract data from the output of a command line for the **get**, **diff**, and **property** operations.

When an operation needs to be performed, the **<Command>**, **<Stdin>**, **<Argument>** and **<Filter>** elements defined for the requested operation are combined together to generate a set of command lines, as explained in the artex_catalog_cmdline_gen.dita#artex_catalog_cmdline_gen/CommandLineGenerationAlgorithm topic. The generated command lines are then run by a shell. For the **get**, **diff**, and **property** operations, the **<Mask>** element is used to extract the requested data (parameter values, comparison results, or property values) from the command output.

Configuration methods

Command line elements can be defined locally inside a **<ParameterDef>** element, or inherited from a **<CfgMethod>** element referenced in the **<ParameterDef>** element using the *cfgmethod* attribute.

Combination are permitted: the set of command line elements defined for a specific operation of a specific parameter is the union of the command line elements defined locally under the **<ParameterDef>** element, and the command line elements defined for the same operation in the **<CfgMethod>** element referenced by the *cfgmethod* attribute of the **<ParameterDef>** element. If the same command line element is defined both locally and in a configuration method, then the local definition takes precedence.

For example, in this non-optimized catalog file:

```
<CfgMethod id=”vmo”>
  <Get type=”nextboot”>
    <Command>/usr/sbin/vmo -r%a</Command>
    <Mask name="1" value="2">[[:space:]]*(.*) = (.*)</Mask>
  </Get>

  <Set type=”permanent”>
    <Command>/usr/sbin/vmo –p –o%a</Command>
    <Argument> -o %n=%p</Argument>
  </Set>

</CfgMethod>

<ParameterDef name=”lgpg_size” cfgmethod=”vmo”>
  <Get type=”current”>
    <Command>/usr/sbin/vmo -o lgpg_size</Command>
```

```
    <Mask name="1" value="2">[[:space:]]*(.*) = (.*)</Mask>
  </Get>

  <Get type=”nextboot”>
    <Argument> -o lgpg_size</Argument>
  </Get>

</ParameterDef>
```

We can see that:

- The **<Get type="current">** operation is entirely defined at the **<ParameterDef>** level.
- The **<Get type="nextboot">** operation has some elements defined at the **<CfgMethod>** level (**<Command>** and **<Mask>**) and some elements defined at the **<ParameterDef>** level (**<Argument>**).
- The **<Get type="current">** operation is entirely defined at the **<CfgMethod>** level.

Using a configuration method offers two main advantages:

- It simplifies the catalog. In many cases parameter definitions will inherit all their command line element from a configuration method, and the **<ParameterDef>** element will be empty.
- It allows different parameters to be grouped together in a single command line, when possible.

Command line generation algorithm

Command lines are generated using an algorithm that allows several parameters to be grouped in a single command.

Parameter grouping is not only desirable from a performance and efficiency standpoint it is also necessary for certain parameters. For example, the **vmo** parameters *lgpg_regions* and *lgpg_size*, which cannot be set independently and need to be set together in a single **vmo** command invocation.

The command line generation algorithm is functionally equivalent to the following steps:

1. Each parameter in the input profile has its **<Command>** and **<Stdin>** elements partially expanded. During this phase, the %a, %v1[name], %v2[name], %f1[name] and %f2[name] sequences are ignored and not expanded.
2. Parameters that verify all five conditions below are grouped together:
   - Parameters use the same **<Command>** element.
   - Parameters use the same **<Stdin>** element.
   - Parameters use the same **<Filter>** element.
   - The expansion of the **<Command>** element performed during step 1 produced identical strings.
   - The expansion of their **<Stdin>** element performed during step 1 produced identical strings.

   The group now has its own, partially expanded **<Command>** and **<Stdin>** elements and its own **<Filter>** element, shared by all the parameters in the group.
3. For each group of parameters, the group **<Command>** and **<Stdin>** elements have the %v1[name], %v2[name], %f1[name] and %f2[name] sequences expanded. Parameter name is only searched within the group.
4. For each group of parameters, the group **<Command>** and **<Stdin>** elements have the %a sequences expanded: each parameter in the group has its **<Argument>** element expanded, and the concatenation of those expanded **<Argument>** elements replaces any %a sequence in the **<Command>** and **<Stdin>** elements.

The result of this process is a set of command lines, with optionally data to write on their standard input and a command to filter their output.

Expansion of command line elements

The **<Command>**, **<Stdin>** and **<Argument>** elements support special sequences that are expanded by the AIX Runtime Expert framework to produce the final command lines.

The table below is a short reference of all the supported sequences. For more details on a sequence, refer to the sections below.

Table 54. Sequence

Sequence	Expands to
%%	The literal % character.
%a	The concatenation of the expanded **Argument** strings for all parameters that can be processed in the same command line.
%n	The name of the parameter.
%v1	The value of the parameter.
%v2	The second value of the parameter. Only valid for **diff** operations.
%f1	The name of the temporary file that will contain the value of the parameter.
%f2	The name of the temporary file that will contain the second value of the parameter. Only valid for **diff** operations.
%v1[name]	The value of parameter name.
%v2[name]	The second value of parameter name. Only valid for **diff** operations.
%f1[name]	The name of a temporary file that will contain the value of the parameter name.
%f2[name]	The name of a temporary file that will contain the second value of parameter name. Only valid for **diff** operations.
%t[class]	The name of the target instance for the target class.
%p[name]	The value of property *name*.
%c	The catalog id.

Escaping of % sequences

Parameter names, parameter values and target names that are expanded by the AIX Runtime Expert are enclosed between single quotes when they are used inside a **<Command>** element or inside an **<Argument>** element that is to be inserted (via the %a sequence) into a **<Command>** element. This is to ensure that those strings will be passed to the shell as a single word, even if they include spaces or other special characters. Additionally, any single quote character within the expanded expression is properly escaped.

The catalog writers must be careful to not use the %n, %v1, %v2, %v1[name], %v2[name] or %t[class] sequences inside a quoted string. If those sequences must be used within a string, the string must be closed before the % sequence as shown in the following example:

```
echo "Parameter "%n" is set to "%v1
```

Failure to do so will result in incorrect command lines and is a security risk.

The %% sequence

The %% sequence expands to the literal % character.

For example, the string:

```
/bin/ps -aeF"%%a"
```

expands to the following string:

```
/bin/ps -aeF"%a"
```

The %a sequence

The %a sequence can be used either in the <Command> string, or in the <Stdin> string. It is substituted with the concatenation of all the expanded <Argument> strings of all the parameters that can be treated in the same command (see the Command line generation topic for a formal description on parameter grouping).

For example, the following catalog (note that it could be simplified by using the %n sequence) :

```
<CfgMethod id=”vmo”>
  <Get type=”current”
    <Command>/usr/sbin/vmo%a</Command>
  </Get>
</CfgMethod>
<ParameterDef name=”lgpg_size” cfgmethod=”vmo”>
  <Get type=”current”>
    <Argument> -o lgpg_size</Argument>
  </Get>
</ParameterDef>
<ParameterDef name=”lgpg_regions” cfgmethod=”vmo”>
  <Get type=”current”>
    <Argument> -o lgpg_regions</Argument>
  </Get>
</ParameterDef>
```

And the following profile:

```
<Parameter name=”lgpg_size” />
<Parameter name=”lgpg_regions” />
```

will produce the following command line for the “get current” operation:

```
/usr/sbin/vmo –o lgpg_size –o lgpg_regions
```

The %n sequence

The %n sequence is substituted with the name of the parameter.

Using the %n sequence, the example from the %a section could be simplified as follows:

```
<CfgMethod id=”vmo”>
<Get type=”current”>
    <Command>/usr/sbin/vmo%a</Command>
    <Argument> -o %n</Argument>
</Get>
</CfgMethod>
<ParameterDef name=”lgpg_size” cfgmethod=”vmo” />
<ParameterDef name=”lgpg_regions” cfgmethod=”vmo” />
```

With the following profile:

```
<Parameter name=”lgpg_size” />
<Parameter name=”lgpg_regions” />
```

The following command line would be generated for the get current operation:

```
/usr/sbin/vmo –o ’lgpg_size’ –o ’lgpg_regions’
```

The %v1 and %v2 sequences

The %v1 sequence is substituted with the value of the parameter.

The %v2 sequence is only valid for **<Diff>** operations and is substituted with the second value of the parameter.

For example, the following catalog:

```
<CfgMethod id=”vmo”>
  <Set type=”permanent”>
    <Command>/usr/sbin/vmo -p%a</Command>
    <Argument> -o %n=%v1</Argument>
  </Set>
</CfgMethod>
<ParameterDef name=”lgpg_size” cfgmethod=”vmo” />
<ParameterDef name=”lgpg_regions” cfgmethod=”vmo” />
```

with the following profile:

```
<Parameter name=”lgpg_size” value=”16M”/>
<Parameter name=”lgpg_regions” value=”128” />
```

would generate the following command line for the **set permanent** operation:

```
/usr/sbin/vmo –p –o ’lgpg_size’=’16M’ –o ’lgpg_regions’=’128’
```

The %f1 and %f2 sequences

The %f1 and %f2 sequences are substituted with the name of temporary file created before the command is executed. The file content is the value of the parameter for %f1 and the second value of the parameter for %f2. The %f2 sequence can only be used for **<Diff>** operations.

For example, the following catalog:

```
<ParameterDef name=”some_file”>
  <Diff>
    <Command>/usr/bin/diff %f1 %f2</Command>
  </Diff>
</ParameterDef>
```

When an **artexdiff** is performed between the two profiles including the same parameter with a different value:

```
<Parameter name=”some_file” value=”foo” />
<Parameter name=”some_file” value=”bar” />
```

Then two temporary files `/tmp/file1` and `/tmp/file2` (actual file names will be different) containing respectively the “foo” and “bar” strings will be created, and the following command will be executed:

```
/usr/bin/diff /tmp/file1 /tmp/file2
```

The %v1[name] and %v2[name] sequences

The %v1[name] sequence is substituted with the value of parameter name.

The %v2[name] sequence is only valid for **<Diff>** operations and is substituted with the second value of parameter name.

Those sequences are useful when a configuration command accepts several parameters at the same time, but require that some of them be placed in a particular position on the command line. This is the case of the **chcons** command for example, which requires that the path to the console device or file come last on the command line. Using the %v1[name] sequence, the **chcons** catalog could be written as follows:

```
<CfgMethod id="chcons">
  <Set type="nextboot">
    <Command>/usr/sbin/chcons%a %v1[console_device]</Command>
    <Argument> -a %n=%v1</Argument>
  </Set>
</CfgMethod>
<ParameterDef name="console_device" cfgmethod="chcons" reboot="true" />
<ParameterDef name="console_logname" cfgmethod="chcons" reboot="true" />
<ParameterDef name="console_logsize" cfgmethod="chcons" reboot="true" />
```

with the following profile:

```
<Parameter name="console_device" value="/dev/vty0"/>
<Parameter name="console_logname" value="/var/adm/ras/conslog" />
<Parameter name="console_logverb" value="9" />
```

This catalog would generate the following command line for the **set nextboot** operation:

```
/usr/sbin/chcons –a 'console_logname'='/var/adm/ras/conslog' –a 'console_logverb'='9' /dev/vty0
```

The %f1[name] and %f2[name] sequences

The %f1[name] and %f2[name] sequences are substituted with the name of temporary file created before the command is executed. The file content is the value of parameter name for %f1[name] and the second value of parameter name for %f2[name]. The %f2[name] sequence can only be used for **<Diff>** operations.

The %t[class] sequences

The %t[class] sequence is substituted with the name of the target instance being treated for target class.

The %t[class] sequence is used for parameters that apply to a specific object, not to the whole system. An example of this is the **chuser** command, whose parameters apply to a specific user (root, guest) for a specific registry (files, LDAP). The catalog for the **chuser** command could be written as follows:

```
<CfgMethod id="chuser">
  <Set type="permanent">
<Command>/usr/bin/chuser –R %t[module]%a %t[user]</Command>
    <Argument> %n=%v1</Argument>
  </Set>
</CfgMethod>
<ParameterDef name="shell" cfgmethod="chuser" targetClass="module,user">
<ParameterDef name="histsize" cfgmethod="chuser" targetClass="module,user" />
```

With the following profile, which sets the shell and *histsize* parameters for users *adam* and *bob* in the LDAP and files registries:

```
<Parameter name="shell" value="/usr/bin/ksh">
  <Target class="module" instance="LDAP" />
  <Target class="user" instance="adam" />
</Parameter>
<Parameter name="histsize" value="5000">
  <Target class="module" instance="LDAP" />
  <Target class="user" instance="adam" />
</Parameter>
<Parameter name="shell" value="/usr/bin/ksh">
  <Target class="module" instance="files" />
  <Target class="user" instance="adam" />
</Parameter>
```

```
<Parameter name="histsize" value="5000">
  <Target class="module" instance="files" />
  <Target class="user" instance="adam" />
</Parameter>
<Parameter name="shell" value="/usr/bin/bash">
  <Target class="module" instance="LDAP" />
  <Target class="user" instance="bob" />
</Parameter>
<Parameter name="histsize" value="10000">
  <Target class="module" instance="LDAP" />
  <Target class="user" instance="bob" />
</Parameter>
<Parameter name="shell" value="/usr/bin/bash">
  <Target class="module" instance="files" />
  <Target class="user" instance="bob" />
</Parameter>
<Parameter name="histsize" value="10000">
  <Target class="module" instance="files" />
  <Target class="user" instance="bob" />
</Parameter>
```

It would execute the following commands:

```
/usr/bin/chuser -R 'LDAP' 'shell'='/usr/bin/ksh' 'histsize'='5000' 'adam'
/usr/bin/chuser -R 'files' 'shell'='/usr/bin/ksh' 'histsize'='5000' 'adam'
/usr/bin/chuser -R 'LDAP' 'shell'='/usr/bin/bash' 'histsize'='10000' 'bob'
/usr/bin/chuser -R 'files' 'shell'='/usr/bin/bash' 'histsize'='10000' 'bob'
```

Notice how four commands were generated. The reason is that the %t[module] and %t[user] sequences were used in the **<Command>** string, meaning that each command is specific to a particular module and user. Because of this, only parameters that apply to the same module and user are grouped together.

The %p[name] sequence

The %p[name] sequence is substituted with the value specified in the input profile for property name. For example, the following prerequisite uses the %p[nodeId] sequence to check that the node id of the local system (returned by the **uname -f** command) matches the node id stored in the nodeId property of the profile:

```
<PrereqDef id="nodeId">
  <Command>[[ `/usr/bin/uname -f` = %p[nodeId] ]]</Command>
  <ErrMessage>Parameter cannot be applied to a different node</ErrMessage>
</PrereqDef>
```

The %c sequence

The %c sequence is substituted with the id of the catalog file that the parameter belongs to. This is the catalog id specified in the profile, which can be different from the id of the catalog that actually defines the parameter if catalog inheritance is used.

For example, the following prerequisite uses the %c sequence to check that the *uniquetype* of the target device matches the name of the catalog file:

```
<PrereqDef id="devUniqueType">
  <Command>[[ "devParam.`/usr/sbin/lsdev -F uniquetype -l %t[device] | /usr/bin/tr / .`" =
%c ]]</Command>
  <ErrMessage>Parameter cannot be applied to a different device type</ErrMessage>
</PrereqDef>
```

Commands and processes

A *command* is a request to perform an operation or run a program. A *process* is a program or command that is actually running on the computer.

You use commands to tell the operating system what task you want it to perform. When commands are entered, they are deciphered by a command interpreter (also known as a *shell*), and that task is processed.

The operating system can run many different processes at the same time.

The operating system allows you to manipulate the input and output (I/O) of data to and from your system by using specific I/O commands and symbols. You can control input by specifying the location from which to gather data. For example, you can specify to read input entered on the keyboard (standard input) or to read input from a file. You can control output by specifying where to display or store data. For example, you can specify to write output data to the screen (standard output) or to write it to a file.

Commands

Some commands can be entered simply by typing one word. It is also possible to combine commands so that the output from one command becomes the input for another command.

Combining commands so that the output from one command becomes the input for another command is known as *piping*.

Flags further define the actions of commands. A *flag* is a modifier used with the command name on the command line, usually preceded by a dash.

Commands can also be grouped together and stored in a file. These files are known as *shell procedures* or *shell scripts*. Instead of executing the commands individually, you execute the file that contains the commands.

To enter a command, type the command name at the prompt, and press Enter.

```
$ CommandName
```

Related concepts

Shell features

There are advantages to using the shell as an interface to the system.

Related tasks

Creating and running a shell script

A *shell script* is a file that contains one or more commands. Shell scripts provide an easy way to carry out tedious commands, large or complicated sequences of commands, and routine tasks. When you enter the name of a shell script file, the system executes the command sequence contained by the file.

Command syntax and command names

Although some commands can be entered by simply typing one word, other commands use flags and parameters. Each command has a syntax that designates both the required and optional flags and parameters.

The general format for a command is as follows:

```
CommandName flag(s) parameter(s)
```

The following are some general rules about commands:

- Spaces between commands, flags, and parameters are significant.
- Two commands can be entered on the same line by separating the commands with a semicolon (;). For example:

  ```
  $ CommandOne;CommandTwo
  ```

  The shell runs the commands sequentially.
- Commands are case-sensitive. The shell distinguishes between uppercase and lowercase letters. To the shell, `print` is not the same as `PRINT` or `Print`.

- A very long command can be entered on more than one line by using the backslash (\) character. A backslash signifies line continuation to the shell. The following example is one command that spans two lines:

```
$ ls Mail info temp \
(press Enter)

> diary
(the > prompt appears)
```

  The > character is your secondary prompt ($ is the nonroot user's default primary prompt), indicating that the current line is the continuation of the previous line. Note that `csh` (the C shell) gives no secondary prompt, and the break must be at a word boundary, and its primary prompt is %.

The first word of every command is the command name. Some commands have only a command name.

Command flags
A number of flags might follow the command name. Flags modify the operation of a command and are sometimes called *options*.

A flag is set off by spaces or tabs and usually starts with a dash (-). Exceptions are **ps**, **tar**, and **ar**, which do not require a dash in front of some of the flags. For example, in the following command:

```
ls -a -F
```

ls is the command name, and **-a -F** are the flags.

When a command uses flags, they come directly after the command name. Single-character flags in a command can be combined with one dash. For example, the previous command can also be written as follows:

```
ls -aF
```

There are some circumstances when a parameter actually begins with a dash (-). In this case, use the delimiter dash dash (–) before the parameter. The – tells the command that whatever follows is not a flag but a parameter.

For example, if you want to create a directory named `-tmp` and you type the following command:

```
mkdir -tmp
```

The system displays an error message similar to the following:

```
mkdir: Not a recognized flag: t
Usage: mkdir [-p] [-m mode] Directory ...
```

The correct way of typing the command is as follows:

```
mkdir -- -tmp
```

Your new directory, `-tmp`, is now created.

Command parameters
After the command name, there might be a number of flags, followed by parameters. Parameters are sometimes called *arguments* or *operands.* Parameters specify information that the command needs in order to run.

If you do not specify a parameter, the command might assume a default value. For example, in the following command:

```
ls -a temp
```

ls is the command name, **-a** is the flag, and *temp* is the parameter. This command displays all (**-a**) the files in the directory `temp`.

In the following example:

```
ls -a
```

the default value is the current directory because no parameter is given.

In the following example:

```
ls temp mail
```

no flags are given, and *temp* and *mail* are parameters. In this case, `temp` and `mail` are two different directory names. The **ls** command displays all but the hidden files in each of these directories.

Whenever a parameter or option-argument is, or contains, a numeric value, the number is interpreted as a decimal integer, unless otherwise specified. Numerals in the range 0 to `INT_MAX`, as defined in the `/usr/include/sys/limits.h` file, are syntactically recognized as numeric values.

If a command you want to use accepts negative numbers as parameters or option-arguments, you can use numerals in the range `INT_MIN` to `INT_MAX`, both as defined in the `/usr/include/sys/limits.h` file. This does not necessarily mean that all numbers within that range are semantically correct. Some commands have a built-in specification permitting a smaller range of numbers, for example, some of the print commands. If an error is generated, the error message lets you know the value is out of the supported range, not that the command is syntactically incorrect.

Usage statements

Usage statements are a way to represent command syntax and consist of symbols such as brackets ([ ]), braces ({ }), and vertical bars (|).

The following is a sample of a usage statement for the **unget** command:

unget [**-r***SID*] [**-s**] [**-n**] *File* ...

The following conventions are used in the command usage statements:

- Items that must be entered literally on the command line are in **bold**. These items include the command name, flags, and literal characters.
- Items representing variables that must be replaced by a name are in *italics*. These items include parameters that follow flags and parameters that the command reads, such as *Files* and *Directories*.
- Parameters enclosed in brackets are optional.
- Parameters enclosed in braces are required.
- Parameters not enclosed in either brackets or braces are required.
- A vertical bar signifies that you choose only one parameter. For example, [a | b] indicates that you *can* choose a, b, or nothing. Similarly, { a | b } indicates that you *must* choose either a or b.
- Ellipses (`...`) signify the parameter can be repeated on the command line.
- The dash (`-`) represents standard input.

Shutdown command

If you have root user authority, you can use the **shutdown** command to stop the system. If you are not authorized to use the **shutdown** command, simply log out of the operating system and leave it running.

Attention: Do not turn off the system without first shutting down. Turning off the system ends all processes running on the system. If other users are working on the system, or if jobs are running in the background, data might be lost. Perform proper shutdown procedures before you stop the system.

At the prompt, type the following:

```
shutdown
```

When the **shutdown** command completes and the operating system stops running, you receive the following message:

```
....Shutdown completed....
```

See the **shutdown** command for the complete syntax.

Locating another command or program (whereis command)

The **whereis** command locates the source, binary, and manuals sections for specified files. The command attempts to find the desired program from a list of standard locations.

See the following examples:

- To find files in the current directory that have no documentation, type the following:

  ```
  whereis -m -u *
  ```

- To find all of the files that contain the name `Mail`, type the following:

  ```
  whereis Mail
  ```

  The system displays information similar to the following:

  ```
  Mail: /usr/bin/Mail /usr/lib/Mail.rc
  ```

Displaying information about a command (man command)

The **man** command displays information on commands, subroutines, and files.

The general format for the **man** command is as follows:

```
man CommandName
```

To obtain information about the **pg** command, type the following:

```
man pg
```

The system displays information similar to the following:

```
  pg Command

  Purpose

  Formats files to the display.

  Syntax

  pg [ - Number ] [ -c ] [ -e ] [ -f ] [ -n ] [ -p String ]
  [ -s ] [ +LineNumber | +/Pattern/ ] [ File ... ]

  Description

  The pg command reads a file name from the File parameter and
  writes the file to standard output one screen at a time. If you
  specify a - (dash) as the File parameter, or run the pg command
  without options, the pg command reads standard input. Each
  screen is followed by a prompt. If you press the Enter key,
  another page is displayed. Subcommands used with the pg command
  let you review or search in the file.
```

Displaying the function of a command (whatis command)

The **whatis** command looks up a given command, system call, library function, or special file name, as specified by the **Command** parameter, from a database you create using the **catman -w** command.

For information about the **catman -w** command, see **catman -w**. The **whatis** command displays the header line from the manual section. You can then issue the **man** command to obtain additional information. For more information about the **man** command, see **man**.

The **whatis** command is equivalent to using the **man -f** command.

To find out what the **ls** command does, type the following:

```
whatis ls
```

The system displays information similar to the following:

```
ls(1)  -Displays the contents of a directory.
```

Listing previously entered commands (history command)

Use the **history** command to list commands that you have previously entered.

The **history** command is a Korn shell built-in command that lists the last 16 commands entered. The Korn shell saves commands that you entered to a command history file, usually named `$HOME/.sh_history`. Using this command saves time when you need to repeat a previous command.

By default, the Korn shell saves the text of the last 128 commands for nonroot users and 512 commands for the root user. The history file size (specified by the *HISTSIZE* environment variable) is not limited, although a very large history file size can cause the Korn shell to start slowly.

Note: The Bourne shell does not support command history.

To list the previous commands you entered, at the prompt, type the following:

```
history
```

The **history** command entered by itself lists the previous 16 commands entered. The system displays information similar to the following:

```
928   ls
929   mail
930   printenv MAILMSG
931   whereis Mail
932   whatis ls
933   cd /usr/include/sys
934   ls
935   man pg
936   cd
937   ls | pg
938   lscons
939   tty
940   ls *.txt
941   printenv MAILMSG
942   pwd
943   history
```

The listing first displays the position of the command in the `$HOME/.sh_history` file followed by the command.

To list the previous five commands, at the prompt, type the following:

```
history -5
```

A listing similar to the following is displayed:

```
939   tty
940   ls *.txt
941   printenv MAILMSG
942   pwd
943   history
944   history -5
```

The **history** command followed by a number lists all the previous commands entered, starting at that number.

To list the commands since 938, at the prompt, type the following:

```
history 938
```

A listing similar to the following is displayed:

```
938   lscons
939   tty
940   ls *.txt
941   printenv MAILMSG
942   pwd
943   history
944   history -5
945   history 938
```

Related concepts

Operating system shells
Your interface to the operating system is called a *shell*.

Command history substitution
Use the **fc** built-in command to list or edit portions of the history file. To select a portion of the file to edit or list, specify the number or the first character or characters of the command.

Repeating commands using the r alias

Use the **r** Korn shell alias to repeat previous commands.

Type `r`, and press Enter, and you can specify the number or the first character or characters of the command.

If you want to list the displays currently available on the system, type `lsdisp` at the prompt. The system returns the information on the screen. If you want the same information returned to you again, at the prompt, type the following:

```
r
```

The system runs the most recently entered command again. In this example, the **lsdisp** command runs.

To repeat the **ls *.txt** command, at the prompt, type the following:

```
r ls
```

The **r** Korn shell alias locates the most recent command that begins with the character or characters specified.

String substitution using the r alias

You can use the **r** Korn shell alias to modify a command before it is run.

In this case, a substitution parameter of the form *Old*=*new* can be used to modify the command before it is run.

The following examples show how to use the **r** alias:

- If command line 940 is **ls** `*.txt`, and you want to run **ls** `*.exe`, at the prompt, type the following:

  ```
  r txt=exe 940
  ```

  This runs command 940, substituting `exe` for `txt`.
- If the command on line 940 is the most recent command that starts with a lowercase letter *l*, you can also type the following:

  ```
  r txt=exe l
  ```

Note: Only the first occurrence of the *Old* string is replaced by the *New* string. Entering the **r** Korn shell alias without a specific command number or character performs the substitution on the immediately previous command entered.

Editing the command history

Use the **fc** Korn shell built-in command to list or edit portions of the command history file.

To select a portion of the file to edit or list, specify the number or the first character or characters of the command. You can specify a single command or range of commands.

If you do not specify an editor program as an argument to the **fc** Korn shell built-in command, the editor specified by the *FCEDIT* variable is used. If the *FCEDIT* variable is not defined, the `/usr/bin/ed` editor is used. The edited command or commands are printed and run when you exit the editor. Use the **printenv** command to display the value of the *FCEDIT* variable.

The following are examples of how to edit the command history:

- If you want to run the command:

  ```
  cd /usr/tmp
  ```

  which is very similar to command line 933, at the prompt, type the following:

  ```
  fc 933
  ```

  At this point, your default editor appears with the command line 933. Change `include/sys` to `tmp`, and when you exit your editor, the edited command is run.
- You can also specify the editor you want to use in the **fc** command. For example, if you want to edit a command using the `/usr/bin/vi` editor, at the prompt, type the following:

  ```
  fc -e vi 933
  ```

  At this point, the `vi` editor appears with the command line 933.
- You can also specify a range of commands to edit. For example, if you want to edit the commands 930 through 940, at the prompt, type the following:

  ```
  fc 930 940
  ```

  At this point, your default editor appears with the command lines 930 through 940. When you exit the editor, all the commands that appear in your editor are run sequentially.

Creating a command alias (alias shell command)

An *alias* lets you create a shortcut name for a command, file name, or any shell text. By using aliases, you save a lot of time when doing tasks you do frequently. You can create a command alias.

Use the **alias** Korn shell built-in command to define a word as an alias for some command. You can use aliases to redefine built-in commands but not to redefine reserved words.

The first character of an alias name can be any printable character except the metacharacters. Any remaining characters must be the same as for a valid file name.

The format for creating an alias is as follows:

```
alias Name=String
```

in which the *Name* parameter specifies the name of the alias, and the *String* parameter specifies a string of characters. If *String* contains blank spaces, enclose it in quotation marks.

The following are examples how to create an alias:

- To create an alias for the command **rm -i** (prompts you before deleting files), at the prompt, type the following:

  ```
  alias rm="/usr/bin/rm -i"
  ```

  In this example, whenever you enter the command **rm**, the actual command performed is `/usr/bin/rm -i`.

- To create an alias named **dir** for the command **ls -alF | pg** (which displays detailed information of all the files in the current directory, including the invisible files; marks executable files with an * and directories with a /; and scrolls per screen), at the prompt, type the following:

```
alias dir="/usr/bin/ls -alF | pg"
```

  In this example, whenever you enter the command **dir**, the actual command performed is `/usr/bin/ls -alF | pg`.
- To display all the aliases you have, at the prompt, type the following:

```
alias
```

  The system displays information similar to the following:

```
rm="/usr/bin/rm -i"
dir="/usr/bin/ls -alF | pg"
```

Related concepts

Command aliasing in the Korn shell or POSIX shell
The Korn shell, or POSIX shell, allows you to create aliases to customize commands.

International character support in text formatting

You can use text formatting commands to work with text composed of the international extended character set used for European languages.

The international extended character set provides the characters and symbols used in many European languages, as well as an ASCII subset composed of English-language characters, digits, and punctuation.

All characters in the European extended character set have ASCII forms. These forms can be used to represent the extended characters in input, or the characters can be entered directly with a device such as a keyboard that supports the European extended characters.

The following text-formatting commands support all international languages that use single-byte characters. These commands are located in `/usr/bin`. (The commands identified with an asterisk (*) support text processing for multibyte languages.

```
addbib*         hyphen          pic*            pstext
checkmm         ibm3812         ps4014          refer*
checknr*        ibm3816         ps630           roffbib*
col*            ibm5587G*       psbanne         soelim*
colcrt          ibm5585H-T*     psdit           sortbib*
deroff*         indxbib*        psplot          tbl*
enscript        lookbib*        psrev           troff*
eqn*            makedev*        psroff          vgrind
grap*           neqn*           psrv            xpreview*
hplj            nroff*
```

Text-formatting commands and macro packages not in the preceding list have not been enabled to process international characters.

Related concepts

Multibyte character support in text formatting
Certain text formatting commands can be used to process text for multibyte languages.

Text formatting with extended single-byte characters

If your input device supports characters from the European-language extended character set, you can enter them directly.

Otherwise, use the following ASCII escape sequence form to represent these characters:

The form \[*N*], where *N* is the 2- or 4-digit hexadecimal code for the character.

> **Note:** The NCesc form \<xx> is no longer supported.

Text containing extended characters is output according to the formatting conventions of the language in use. Characters that are not defined for the interface to a specific output device produce no output or error indication.

Although the names of the requests, macro packages, and commands are based on English, most of them can accept input (such as file names and parameters) containing characters in the European extended character set.

For the **nroff** and **troff** commands and their preprocessors, the command input must be ASCII, or an unrecoverable syntax error will result. International characters, either single-byte or multibyte, can be entered when enclosed within quotation marks and within other text to be formatted. For example, using macros from the **pic** command:

```
define foobar % SomeText %
```

After the `define` directive, the specified name, `foobar`, must be ASCII. However, the replacement text, `SomeText`, can contain non-ASCII characters.

Multibyte character support in text formatting

Certain text formatting commands can be used to process text for multibyte languages.

These commands are identified with an asterisk (*) in the list under International character support in text formatting. Text formatting commands not in the list have not been enabled to process international characters.

If supported by your input device, multibyte characters can be entered directly. Otherwise, you can enter any multibyte character in the ASCII form \[*N*], where *N* is the 2-, 4-, 6-, 7-, or 8-digit hexadecimal encoding for the character.

Although the names of the requests, macros, and commands are based on English, most of them can accept input (such as file names and parameters) containing any type of multibyte character.

If you are already familiar with using text-formatting commands with single-byte text, the following list summarizes characteristics that are noteworthy or unique to the multibyte locales:

- Text is not hyphenated.
- Special format types are required for multibyte numerical output. Japanese format types are available.
- Text is output in horizontal lines, filled from left to right.
- Character spacing is constant, so characters automatically align in columns.
- Characters that are not defined for the interface to a specific output device produce no output or error indication.

Related concepts

International character support in text formatting
You can use text formatting commands to work with text composed of the international extended character set used for European languages.

Displaying a Calendar

You can write a calendar to standard output by using the **cal** command.

The **Month** parameter names the month for which you want the calendar. It can be a number from 1 through 12 for January through December, respectively. If no **Month** is specified, the **cal** command defaults to the current month.

The **Year** parameter names the year for which you want the calendar. Because the **cal** command can display a calendar for any year from 1 through 9999, type the full year rather than just the last two digits. If no **Year** is specified, the **cal** command defaults to the present year.

The following are examples of how to use the **cal** command:

1. To display a calendar for February 2002 at your workstation, type:

```
cal 2 2002
```

2. Press Enter.
3. To print a calendar for the year 2002, type:

```
cal 2002 | qprt
```

4. Press Enter.

Displaying reminder messages

You can display a reminder message by reading a file named **calendar**. This file is created in your home directory with the **calendar** command. The command writes to standard output any line in the file that contains today's or tomorrow's date.

You can read a file named `calendar`, which you create in your home directory with the **filepath** command. The command writes to standard output any line in the file that contains today's or tomorrow's date.

The **calendar** command recognizes date formats such as Dec. 7 or 12/7. It also recognizes the special character asterisk (*) when it is followed by a slash (/). It interprets */7, for example, as signifying the seventh day of every month.

On Fridays, the **calendar** command writes all lines containing the dates for Friday, Saturday, Sunday, and Monday. The command does not, however, recognize holidays. On holidays the command functions as usual and gives only the next day's schedule.

Using a typical calendar file

A typical `calendar` file might look similar to the following:

```
*/25 - Prepare monthly report
Aug. 12 - Fly to Denver
aug 23 - board meeting
Martha out of town - 8/23, 8/24, 8/25
8/24 - Mail car payment
sat aug/25 - beach trip
August 27 - Meet with Simmons
August 28 - Meet with Wilson
```

To run the **calendar** command, type:

```
calendar
```

If today is Friday, August 24, the **calendar** command displays the following:

```
*/25 - Prepare monthly report
Martha out of town - 8/23, 8/24, 8/25
8/24 - Mail car payment
sat aug/25 - beach trip
August 27 - Meet with Simmons
```

Using a calendar file that contains an include statement

A `calendar` file that contains an include statement might look like the following:

```
#include </tmp/out>
1/21 -Annual review
1/21 -Weekly project meeting
1/22 *Meet with Harrison in Dallas*
Doctor's appointment - 1/23
1/23 -Vinh's wedding
```

To run the **calendar** command, type:

```
calendar
```

If today is Wednesday, January 21, the **calendar** command displays the following:

```
Jan.21 Goodbye party for David
Jan.22 Stockholder meeting in New York
1/21 -Annual review
1/21 -Weekly project meeting
1/22 *Meet with Harrison in Dallas*
```

The results of the **calendar** command indicate the /tmp/out file contained the following lines:

```
Jan.21 Goodbye party for David
Jan.22 Stockholder meeting in New York
```

Factoring a Number

You can factor numbers with the **factor** command.

When called without specifying a value for the **Number** parameter, the **factor** command waits for you to enter a positive number less than 1E14 (100,000,000,000,000). It then writes the prime factors of that number to standard output. It displays each factor in order and the proper number of times if the same factor is used more than once. To exit, enter 0 (zero) or any non-numeric character.

When called with an argument, the **factor** command determines the prime factors of the **Number** parameter, writes the results to standard output, and exits.

The following is an example of how to calculate factors:

1. To calculate the prime factors of the number 123, type:

   ```
   factor 123
   ```

2. Press Enter. The following displays:

   ```
   123 3 41
   ```

Locating a command by keyword

You can display the man page sections that contain any of the given *Keywords* in their title by using the **apropos** command.

The **apropos** command considers each word separately is not case-sensitive. Words that are part of other words are also displayed. For example, when looking for the word *compile*, the **apropos** command also finds all instances of the word *compiler*.

Note: The database containing the keywords is /usr/share/man/whatis, which must first be generated with the **catman -w** command.

The **apropos** command is equivalent to using the **man** command with the **-k** option.

For example, to find the manual sections that contain the word *password* in their titles, run the following command:

```
apropos password
```

Processes

A program or command that is actually running on the computer is referred to as a *process*.

Processes exist in parent-child hierarchies. A process started by a program or command is a *parent process*; a *child process* is the product of the parent process. A parent process can have several child processes, but a child process can have only one parent.

The system assigns a process identification number (PID number) to each process when it starts. If you start the same program several times, it will have a different PID number each time.

When a process is started on a system, the process uses a part of the available system resources. When more than one process is running, a scheduler that is built into the operating system gives each process

a share of the computer's time, based on established priorities. These priorities can be changed by using the **nice** or **renice** commands.

Note: To change a process priority to a higher one, you must have root user authority. All users can lower priorities on a process they start by using the **nice** command or on a process they have already started, by using the **renice** command.

The following list describes the types of processes:

Foreground and background processes

Processes that require a user to start them or to interact with them are called *foreground processes*. Processes that are run independently of a user are referred to as *background processes*. Programs and commands run as foreground processes by default. To run a process in the background, place an ampersand (&) at the end of the command name that you use to start the process.

Daemon processes

Daemons are processes that run unattended. They are constantly in the background and are available at all times. Daemons are usually started when the system starts, and they run until the system stops. A daemon process typically performs system services and is available at all times to more than one task or user. Daemon processes are started by the root user or root shell and can be stopped only by the root user. For example, the **qdaemon** process provides access to system resources such as printers. Another common daemon is the **sendmail** daemon.

Zombie processes

A *zombie process* is a dead process that is no longer executing but is still recognized in the process table (in other words, it has a PID number). It has no other system space allocated to it. Zombie processes have been killed or have exited and continue to exist in the process table until the parent process dies or the system is shut down and restarted. Zombie processes display as `<defunct>` when listed by the **ps** command.

Process startup

You start a foreground process from a display station by either entering a program name or command name at the system prompt.

After a foreground process has started, the process interacts with you at your display station until it is complete. No other interaction (for example, entering another command) can take place at the display station until the process is finished or you halt it.

A single user can run more than one process at a time, up to a default maximum of 40 processes per user.

Starting a process in the foreground

To start a process in the foreground, enter the name of the command with the appropriate parameters and flags:

```
$ CommandName
```

Starting a process in the background

To run a process in the background, type the name of the command with the appropriate parameters and flags, followed by an ampersand (&):

```
$ CommandName&
```

When a process is running in the background, you can perform additional tasks by entering other commands at your display station.

Generally, background processes are most useful for commands that take a long time to run. However, because they increase the total amount of work the processor is doing, background processes can slow down the rest of the system.

Most processes direct their output to standard output, even when they run in the background. Unless redirected, standard output goes to the display device. Because the output from a background

process can interfere with your other work on the system, it is usually good practice to redirect the output of a background process to a file or a printer. You can then look at the output whenever you are ready.

Note: Under certain circumstances, a process might generate its output in a different sequence when run in the background than when run in the foreground. Programmers might want to use the **fflush** subroutine to ensure that output occurs in the correct order regardless of whether the process runs in foreground or background.

While a background process is running, you can check its status with the **ps** command.

Command to check the process status (ps command)

Any time the system is running, processes are also running. You can use the **ps** command to find out which processes are running and display information about those processes.

The **ps** command has several flags that enable you to specify which processes to list and what information to display about each process.

To show all processes running on your system, at the prompt, type the following:

```
ps -ef
```

The system displays information similar to the following:

```
    USER   PID  PPID   C    STIME    TTY  TIME CMD
    root     1     0   0   Jun 28      -  3:23 /etc/init
    root  1588  6963   0   Jun 28      -  0:02 /usr/etc/biod 6
    root  2280     1   0   Jun 28      -  1:39 /etc/syncd 60
    mary  2413 16998   2 07:57:30      -  0:05 aixterm
    mary 11632 16998   0 07:57:31  lft/1  0:01 xbiff
    mary 16260  2413   1 07:57:35  pts/1  0:00 /bin/ksh
    mary 16469     1   0 07:57:12  lft/1  0:00 ksh /usr/lpp/X11/bin/xinit
    mary 19402 16260  20 09:37:21  pts/1  0:00 ps -ef
```

The columns in the previous output are defined as follows:

Item	Description
USER	User login name
PID	Process ID
PPID	Parent process ID
C	CPU utilization of process
STIME	Start time of process
TTY	Controlling workstation for the process
TIME	Total execution time for the process
CMD	Command

In the previous example, the process ID for the **ps -ef** command is 19402. Its parent process ID is 16260, the /bin/ksh command.

If the listing is very long, the top portion scrolls off the screen. To display the listing one page (screen) at a time, pipe the **ps** command to the **pg** command. At the prompt, type the following:

```
ps -ef | pg
```

To display status information of all processes running on your system, at the prompt, type the following:

```
ps gv
```

This form of the command lists a number of statistics for each active process. Output from this command looks similar to the following:

```
  PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
    0      - A     0:44    7     8     8    xx     0     0  0.0  0.0 swapper
    1      - A     1:29  518   244   140    xx    21    24  0.1  1.0 /etc/init
  771      - A     1:22    0    16    16    xx     0     0  0.0  0.0 kproc
 1028      - A     0:00   10    16     8    xx     0     0  0.0  0.0 kproc
 1503      - A     0:33  127    16     8    xx     0     0  0.0  0.0 kproc
 1679      - A     1:03  282   192    12 32768   130     0  0.7  0.0 pcidossvr
 2089      - A     0:22  918    72    28    xx     1     4  0.0  0.0 /etc/sync
 2784      - A     0:00    9    16     8    xx     0     0  0.0  0.0 kproc
 2816      - A     5:59 6436  2664   616     8   852   156  0.4  4.0 /usr/lpp/
 3115      - A     0:27  955   264   128    xx    39    36  0.0  1.0 /usr/lib/
 3451      - A     0:00    0    16     8    xx     0     0  0.0  0.0 kproc
 3812      - A     0:00   21   128    12 32768    34     0  0.0  0.0 usr/lib/lpd/
 3970      - A     0:00    0    16     8    xx     0     0  0.0  0.0 kproc
 4267      - A     0:01  169   132    72 32768    16    16  0.0  0.0 /etc/sysl
 4514  lft/0 A     0:00   60   200    72    xx    39    60  0.0  0.0 /etc/gett
 4776  pts/3 A     0:02  250   108   280     8   303   268  0.0  2.0 -ksh
 5050      - A     0:09 1200   424   132 32768   243    56  0.0  1.0 /usr/sbin
 5322      - A     0:27 1299   156   192    xx    24    24  0.0  1.0 /etc/cron
 5590      - A     0:00    2   100    12 32768    11     0  0.0  0.0 /etc/writ
 5749      - A     0:00    0   208    12    xx    13     0  0.0  0.0 /usr/lpp/
 6111      - T     0:00   66   108    12 32768    47     0  0.0  0.0 /usr/lpp/
```

Setting the initial priority of a process (nice command)

You can set the initial priority of a process to a value lower than the base scheduling priority.

To set the initial priority of a process to a value lower than the base scheduling priority, use the **nice** command to start the process.

Note: To run a process at a higher priority than the base scheduling priority, you must have root user authority.

To set the initial priority of a process, type the following:

```
nice -n Number CommandString
```

where *Number* is in the range of 0 to 39, with 39 being the lowest priority. The *nice value* is the decimal value of the system-scheduling priority of a process. The higher the number, the lower the priority. If you use zero, the process will run at its base scheduling priority. *CommandString* is the command and flags and parameters you want to run.

You can also use the **smit nice** command to perform this task.

Changing the priority of a running process (renice command)

You can change the scheduling priority of a running process to a value lower or higher than the base scheduling priority by using the **renice** command from the command line. This command changes the nice value of a process.

Note: To run a process at a higher priority or to change the priority for a process that you did not start, you must have root user authority.

To change the priority of a running process, type the following:

```
renice Priority -p ProcessID
```

where *Priority* is a number in the range of -20 to 20. The higher the number, the lower the priority. If you use zero, the process will run at its base scheduling priority. *ProcessID* is the PID for which you want to change the priority.

You can also use the **smit renice** command to perform this task.

Foreground process cancellation

If you start a foreground process and then decide that you do not want it to finish, you can cancel it by pressing INTERRUPT. This is usually Ctrl-C or Ctrl-Backspace.

Note: INTERRUPT (Ctrl-C) does not cancel background processes. To cancel a background process, you must use the **kill** command.

Most simple commands run so quickly that they finish before you have time to cancel them. The examples in this section, therefore, use a command that takes more than a few seconds to run: **find / -type f**. This command displays the path names for all files on your system. You do not need to study the **find** command in order to complete this section; it is used here simply to demonstrate how to work with processes.

In the following example, the **find** command starts a process. After the process runs for a few seconds, you can cancel it by pressing the INTERRUPT key:

```
$ find / -type f
/usr/sbin/acct/lastlogin
/usr/sbin/acct/prctmp
/usr/sbin/acct/prdaily
/usr/sbin/acct/runacct
/usr/sbin/acct/sdisk
/usr/sbin/acct/shutacct INTERRUPT (Ctrl-C)
$ _
```

The system returns the prompt to the screen. Now you can enter another command.

Related tasks

List of control key assignments for your terminal (stty command)
To display your terminal settings, use the **stty** command. Note especially which keys your terminal uses for control keys.

Keyboard command to stop a foreground process

It is possible for a process to be stopped but not have its process ID (PID) removed from the process table. You can stop a foreground process by pressing Ctrl-Z from the keyboard.

Note: Ctrl-Z works in the Korn shell (**ksh**) and C shell (**csh**), but not in the Bourne shell (**bsh**).

Restarting a stopped process

This procedure describes how to restart a process that has been stopped with a Ctrl-Z.

Note: Ctrl-Z works in the Korn shell (**ksh**) and C shell (**csh**), but not in the Bourne shell (**bsh**). To restart a stopped process, you must either be the user who started the process or have root user authority.

1. To show all the processes running or stopped but not those killed on your system, type the following:

   ```
   ps -ef
   ```

   You might want to pipe this command through a **grep** command to restrict the list to those processes most likely to be the one you want to restart. For example, if you want to restart a **vi** session, you could type the following:

   ```
   ps -ef | grep vi
   ```

   This command would display only those lines from the **ps** command output that contained the word `vi`. The output would look something like this:

   ```
   UID    PID   PPID   C      STIME       TTY  TIME  COMMAND
   root   1234  13682  0      00:59:53    -    0:01  vi test
   root  14277  13682  1      01:00:34    -    0:00  grep vi
   ```

2. In the **ps** command output, find the process you want to restart and note its PID number. In the example, the PID is 1234.
3. To send the CONTINUE signal to the stopped process, type the following:

   ```
   kill -19 1234
   ```

   Substitute the PID of your process for the 1234. The `-19` indicates the CONTINUE signal. This command restarts the process in the background. If the process can run in the background, you are finished with the procedure. If the process must run in the foreground (as a **vi** session would), you must proceed with the next step.

4. To bring the process in to the foreground, type the following:

   ```
   fg 1234
   ```

   Once again, substitute the PID of your process for the 1234. Your process should now be running in the foreground. (You are now in your **vi** edit session).

Scheduling a process for later operation

You can set up a process as a *batch process* to run in the background at a scheduled time.

The **at** and **smit** commands let you enter the names of commands to be run at a later time and allow you to specify when the commands should be run.

Note: The `/var/adm/cron/at.allow` and `/var/adm/cron/at.deny` files control whether you can use the **at** command. A person with root user authority can create, edit, or delete these files. Entries in these files are user login names with one name to a line. The following is an example of an `at.allow` file:

```
root
nick
dee
sarah
```

If the `at.allow` file exists, only users whose login names are listed in it can use the **at** command. A system administrator can explicitly stop a user from using the **at** command by listing the user's login name, in the `at.deny` file. If only the `at.deny` file exists, any user whose name does not appear in the file can use the **at** command.

You cannot use the **at** command if any one of the following is true:

- The `at.allow` file and the `at.deny` file do not exist (allows root user only).
- The `at.allow` file exists but the user's login name is not listed in it.
- The `at.deny` file exists and the user's login name is listed in it.

If the `at.allow` file does not exist and the `at.deny` file does not exist or is empty, only someone with root user authority can submit a job with the **at** command.

The **at** command syntax allows you to specify a date string, a time and day string, or an increment string for when you want the process to run. It also allows you to specify which shell or queue to use. The following examples show some typical uses of the command.

For example, if your login name is `joyce` and you have a script named `WorkReport` that you want to run at midnight, do the following:

1. Type the time you want the program to start running:

   ```
   at midnight
   ```

2. Type the names of the programs to run, pressing Enter after each name. After typing the surname, press the end-of-file character (Ctrl-D) to signal the end of the list.

   ```
   WorkReport^D
   ```

   After you press Ctrl-D, the system displays information similar to the following:

   ```
   job joyce.741502800.a at Fri Jul  6 00:00:00 CDT 2002.
   ```

   The program `WorkReport` is given the job number `joyce.741502800.a` and will run at midnight, July 6.

3. To list the programs you have sent to be run later, type the following:

   ```
   at -l
   ```

   The system displays information similar to the following:

```
joyce.741502800.a       Fri Jul  6 00:00:00 CDT 2002
```

See the **at** command for the complete syntax.

Related tasks

Listing all scheduled processes (at or atq command)
Use the **-l** flag with the **at** command or with the **atq** command to list all scheduled processes.

Removing a process from the schedule
You can remove a scheduled process with the **at** command using the **-r** flag.

Listing all scheduled processes (at or atq command)

Use the **-l** flag with the **at** command or with the **atq** command to list all scheduled processes.

Both commands give the same output; however, the **atq** command can order the processes in the same amount of time that the **at** command is issued and displays only the number of processes in the queue.

You can list all scheduled processes in the following ways:

- With the **at** command from the command line
- With the **atq** command

at command

To list the scheduled processes, type the following:

```
at -l
```

This command lists all the scheduled processes in your queue. If you are a root user, this command lists all the scheduled processes for all users. For complete details of the syntax, see the **at** command.

atq command

See the following examples on how to use the **atq** command:

- To list all scheduled processes in the queue, type the following:

  ```
  atq
  ```

- If you are a root user, you can list the scheduled processes in a particular user's queue by typing:

  ```
  atq UserName
  ```

- To list the number of scheduled processes in the queue, type the following:

  ```
  atq -n
  ```

Related tasks

Scheduling a process for later operation
You can set up a process as a *batch process* to run in the background at a scheduled time.

Removing a process from the schedule
You can remove a scheduled process with the **at** command using the **-r** flag.

Removing a process from the schedule

You can remove a scheduled process with the **at** command using the **-r** flag.

See the following example on how to use the **at** or **atq** command:

1. To remove a scheduled process, you must know its process number.
   You can obtain the process number by using the **at -l** command or the **atq** command.
2. When you know the number of the process you want to remove, type the following:

```
at -r ProcessNumber
```

You can also use the `smit rmat` command to perform this task.

Related tasks

Listing all scheduled processes (at or atq command)
Use the **-l** flag with the **at** command or with the **atq** command to list all scheduled processes.

Scheduling a process for later operation
You can set up a process as a *batch process* to run in the background at a scheduled time.

Removing a background process (kill command)

If INTERRUPT does not halt your foreground process or if you decide, after starting a background process, that you do not want the process to finish, you can cancel the process with the **kill** command.

Before you can cancel a process using the **kill** command, you must know its PID number. The general format for the **kill** command is as follows:

```
kill ProcessID
```

Note:

- To remove a process, you must have root user authority or be the user who started the process. The default signal to a process from the **kill** command is -15 (SIGTERM).
- To remove a zombie process, you must remove its parent process.

1. Use the **ps** command to determine the process ID of the process you want to remove. You might want to pipe this command through a **grep** command to list only the process you want. For example, if you want the process ID of a vi session, you could type the following:

   ```
   ps -l | grep vi
   ```

2. In the following example, you issue the **find** command to run in the background. You then decide to cancel the process. Issue the **ps** command to list the PID numbers.

   ```
   $ find / -type f > dir.paths &
   [1]    21593
   $ ps
      PID    TTY  TIME  COMMAND
     1627  pts3  0:00  ps
     5461  pts3  0:00  ksh
    17565  pts3  0:00  -ksh
    21593  pts3  0:00  find / -type f
   $ kill 21593
   $ ps
      PID    TTY  TIME  COMMAND
     1627  pts3  0:00  ps
     5461  pts3  0:00  ksh
    17565  pts3  0:00  -ksh
   [1]  +  Terminated 21593     find  /  -type f > dir.paths  &
   ```

   The command **kill 21593** ends the background `find` process, and the second **ps** command returns no status information about PID 21593. The system does not display the termination message until you enter your next command, unless that command is **cd**.

   The **kill** command lets you cancel background processes. You might want to do this if you realize that you have mistakenly put a process in the background or that a process is taking too long to run.

The **kill** command can also used in **smit** by typing:

```
smit kill
```

Command summary for commands and processes

The following are commands for commands and processes.

Table 55. Command summary for commands

Item	Description
alias	Shell command that prints a list of aliases to standard output
history	Shell command that displays the history event list
man	Displays information about commands, subroutines, and files online
whatis	Describes the function a command performs
whereis	Locates the source, binary, or manual for installed programs

Table 56. Command summary for processes

Item	Description
at	Runs commands at a later time, lists all scheduled processes, or removes a process from the schedule
atq	Displays the queue of jobs waiting to be run
kill	Sends a signal to running processes
nice	Runs a command at a lower or higher priority
ps	Shows current status of processes
renice	Alters priority of running processes

Managing system hang

System hang management allows users to run mission-critical applications continuously while improving application availability. System hang detection alerts the system administrator of possible problems and then allows the administrator to log in as root or to reboot the system to resolve the problem.

shconf command

The **shconf** command is invoked when **System Hang Detection** is enabled. The **shconf** command configures which events are surveyed and what actions are to be taken if such events occur. You can specify any of the following actions, the priority level to check, the time out while no process or thread executes at a lower or equal priority, the terminal device for the warning action, and the **getty** command action:

- Log an error in `errlog` file
- Display a warning message on the system console (alphanumeric console) or on a specified TTY
- Reboot the system
- Give a special **getty** to allow the user to log in as root and launch commands
- Launch a command

For the **Launch a command** and **Give a special getty** options, system hang detection launches the special **getty** command or the specified command at the highest priority. The special **getty** command prints a warning message that it is a recovering **getty** running at priority 0. The following table captures the various actions and the associated default parameters for priority hang detection. Only one action is enabled for each type of detection.

Option	Enablement	Priority	Timeout (seconds)
Log an error in `errlog` file	disabled	60	120

Option	Enablement	Priority	Timeout (seconds)
Display a warning message	disabled	60	120
Give a recovering getty	enabled	60	120
Launch a command	disabled	60	120
Reboot the system	disabled	39	300

Note: When **Launch a recovering getty on a console** is enabled, the **shconf** command adds the **-u** flag to the **getty** command in the **inittab** that is associated with the console login.

For lost IO detection, you can set the time out value and enable the following actions:

Option	Enablement
Display a warning message	disabled
Reboot the system	disabled

shdaemon daemon

The **shdaemon** daemon is a process that is launched by **init** and runs at priority 0 (zero). It is in charge of handling system hang detection by retrieving configuration information, initiating working structures, and starting detection times set by the user.

Related concepts

Priority hang detection
AIX can detect system hang conditions and try to recover from such situations, based on user-defined actions.

Lost I/O hang detection
AIX can detect system hang conditions and try to recover from such situations, based on user-defined actions.

Configuring system hang detection

You can manage the system hang detection configuration from the SMIT management tool.

SMIT menu options allow you to enable or disable the detection mechanism, display the current state of the feature, and change or show the current configuration. The fast paths for system hang detection menus are:

smit shd
Manage System Hang Detection

smit shstatus
System Hang Detection Status

smit shpriocfg
Change/Show Characteristics of Priority Problem Detection

smit shreset
Restore Default Priority Problem Configuration

smit shliocfg
Change/Show Characteristics of Lost I/O Detection

smit shlioreset
Restore Default Lost I/O Detection Configuration

You can also manage system hang detection using the **shconf** command.

Priority hang detection

AIX can detect system hang conditions and try to recover from such situations, based on user-defined actions.

All processes (also known as threads) run at a priority. This priority is numerically inverted in the range 0-126. Zero is highest priority and 126 is the lowest priority. The default priority for all threads is 60. The priority of a process can be lowered by any user with the **nice** command. Anyone with root authority can also raise a process's priority.

The kernel scheduler always picks the highest priority runnable thread to put on a CPU. It is therefore possible for a sufficient number of high priority threads to completely tie up the machine such that low priority threads can never run. If the running threads are at a priority higher than the default of 60, this can lock out all normal shells and logins to the point where the system appears hung.

The System Hang Detection feature provides a mechanism to detect this situation and allow the system administrator a means to recover. This feature is implemented as a daemon (**shdaemon**) that runs at the highest process priority. This daemon queries the kernel for the lowest priority thread run over a specified interval. If the priority is above a configured threshold, the daemon can take one of several actions. Each of these actions can be independently enabled, and each can be configured to trigger at any priority and over any time interval. The actions and their defaults are:

```
    Action           Default    Default     Default   Default
                     Enabled    Priority    Timeout   Device

1)  Log an error        no         60          2
2)  Console message     no         60          2      /dev/console
3)  High priority       yes        60          2      /dev/tty0
    login shell
4)  Run a command at    no         60          2
    high priority
5)  Crash and reboot    no         39          5
```

Related concepts

Managing system hang
System hang management allows users to run mission-critical applications continuously while improving application availability. System hang detection alerts the system administrator of possible problems and then allows the administrator to log in as root or to reboot the system to resolve the problem.

Lost I/O hang detection

AIX can detect system hang conditions and try to recover from such situations, based on user-defined actions.

Because of I/O errors, the I/O path can become blocked and further I/O on that path is affected. In these circumstances it is essential that the operating system alert the user and execute user defined actions. As part of the Lost I/O detection and notification, the **shdaemon**, with the help of the Logical Volume Manager, monitors the I/O buffers over a period of time and checks whether any I/O is pending for too long a period of time. If the wait time exceeds the threshold wait time defined by the **shconf** file, a lost I/O is detected and further actions are taken. The information about the lost I/O is documented in the error log. Also based on the settings in the **shconf** file, the system might be rebooted to recover from the lost I/O situation.

For lost I/O detection, you can set the time out value and also enable the following actions:

Action	Default Enabled	Default Device
Console message	no	/dev/console
Crash and reboot	no	-

For more information on system hang detection, see “Managing system hang ” on page 140.

Related concepts

Managing system hang

System hang management allows users to run mission-critical applications continuously while improving application availability. System hang detection alerts the system administrator of possible problems and then allows the administrator to log in as root or to reboot the system to resolve the problem.

Process management

The process is the entity that the operating system uses to control the use of system resources. *Threads* can control processor-time consumption, but most system management tools still require you to refer to the process in which a thread is running, rather than to the thread itself.

Tools are available to:

- Observe the creation, cancellation, identity, and resource consumption of processes
  - The **ps** command is used to report process IDs, users, CPU-time consumption, and other attributes.
  - The **who -u** command reports the shell process ID of logged-on users.
  - The **svmon** command is used to report process real-memory consumption.
  - The **acct** command mechanism writes records at process termination summarizing the process's resource use.
- Control the priority level at which a process contends for the CPU.
  - The **nice** command causes a command to be run with a specified process priority.
  - The **renice** command changes the priority of a given process.
- Terminate processes that are out of control.
  - The **kill** command sends a termination signal to one or more processes.

Process monitoring

You, as the system administrator, can manage processes.

The **ps** command is the primary tool for observing the processes in the system. Most of the flags of the **ps** command fall into one of two categories:

- Flags that specify which types of processes to include in the output
- Flags that specify which attributes of those processes are to be displayed

The most widely useful variants of **ps** for system-management purposes are:

Item	Description
ps -ef	Lists all nonkernel processes, with the userid, process ID, recent CPU usage, total CPU usage, and the command that started the process (including its parameters).
ps -fu *UserID*	Lists all of the processes owned by *UserID*, with the process ID, recent CPU usage, total CPU usage, and the command that started the process (including its parameters).

To identify the current heaviest users of CPU time, you could enter:

```
ps -ef | egrep -v "STIME|$LOGNAME" | sort +3 -r | head -n 15
```

This command lists, in descending order, the 15 most CPU-intensive processes other than those owned by you.

For more specialized uses, the following two tables are intended to simplify the task of choosing **ps** flags by summarizing the effects of the flags.

Process-Specifying Flags													
	-A	**-a**	**-d**	**-e**	**-G -g**	**-k**	**-p**	**-t**	**-U -u**	**a**	**g**	**t**	**x**
All processes	Y	-	-	-	-	-	-	-	-	-	Y	-	-
Not processes group leaders and not associated with a terminal	-	Y	-	-	-	-	-	-	-	-	-	-	-
Not process group leaders	-	-	Y	-	-	-	-	-	-	-	-	-	-
Not kernel processes	-	-	-	Y	-	-	-	-	-	-	-	-	-
Members of specified-process groups	-	-	-	-	Y	-	-	-	-	-	-	-	-
Kernel processes	-	-	-	-	-	Y	-	-	-	-	-	-	-
Those specified in process number list	-	-	-	-	-	-	Y	-	-	-	-	-	-
Those associated with tty(s) in the list	-	-	-	-	-	-	-	Y (*n* ttys)	-	-	-	Y (1 tty)	-
Specified user processes	-	-	-	-	-	-	-	-	Y	-	-	-	-
Processes with terminals	-	-	-	-	-	-	-	-	-	Y	-	-	-
Not associated with a tty	-	-	-	-	-	-	-	-	-	-	-	-	Y

Column-Selecting Flags										
Default1	**-f**	**-l**	**-U -u**	**Default2**	**e**	**l**	**s**	**u**	**v**	
PID	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
TTY	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
TIME	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Column-Selecting Flags *(continued)*										
Default1	**-f**	**-l**	**-U -u**	**Default2**	**e**	**l**	**s**	**u**	**v**	
CMD	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
USER	-	Y	-	-	-	-	-	-	Y	-
UID	-	-	Y	Y	-	-	Y	-	-	-
PPID	-	Y	Y	-	-	-	Y	-	-	-
C	-	Y	Y	-	-	-	Y	-	-	-
STIME	-	Y	-	-	-	-	-	-	Y	-
F	-	-	Y	-	-	-	-	-	-	-
S/STAT	-	-	Y	-	Y	Y	Y	Y	Y	Y
PIR	-	-	Y	-	-	-	Y	-	-	-
NI/NICE	-	-	Y	-	-	-	Y	-	-	-
ADDR	-	-	Y	-	-	-	Y	-	-	-
SIZE	-	-	-	-	-	-	-	-	Y	-
SZ	-	Y	-	-	-	Y	-	Y	-	-
WCHAN	-	-	Y	-	-	-	Y	-	-	-
RSS	-	-	-	-	-	-	Y	-	Y	Y
SSIZ	-	-	-	-	-	-	-	Y	-	-
%CPU	-	-	-	-	-	-	-	-	Y	Y
%MEM	-	-	-	-	-	-	-	-	Y	Y
PGIN	-	-	-	-	-	-	-	-	-	Y
LIM	-	-	-	-	-	-	-	-	-	Y
TSIZ	-	-	-	-	-	-	-	-	-	Y
TRS	-	-	-	-	-	-	-	-	-	Y
Environment (following the command)	-	-	-	-	-	Y	-	-	-	-

If **ps** is given with no flags or with a process-specifying flag that begins with a minus sign, the columns displayed are those shown for Default1. If the command is given with a process-specifying flag that does not begin with minus, Default2 columns are displayed. The **-u** or **-U** flag is both a process-specifying and column-selecting flag.

The following are brief descriptions of the contents of the columns:

Item	Description
PID	Process ID
TTY	Terminal or pseudo-terminal associated with the process
TIME	Cumulative CPU time consumed, in minutes and seconds
CMD	Command the process is running
USER	Login name of the user to whom the process belongs

Item	Description
UID	Numeric user ID of the user to whom the process belongs
PPID	ID of the parent process of this process
C	Recently used CPU time
STIME	Time the process started, if less than 24 hours. Otherwise the date the process is started
F	Eight-character hexadecimal value describing the flags associated with the process (see the detailed description of the **ps** command)
S/STAT	Status of the process (see the detailed description of the **ps** command)
PRI	Current priority value of the process
NI/NICE	Nice value for the process
ADDR	Segment number of the process stack
SIZE	(**-v** flag) The virtual size of the data section of the process (in kilobytes)
SZ	(**-l** and **l** flags) The size in kilobytes of the core image of the process.
WCHAN	Event on which the process is waiting
RSS	Sum of the numbers of working-segment and code-segment pages in memory times 4
SSIZ	Size of the kernel stack
%CPU	Percentage of time since the process started that it was using the CPU
%MEM	Nominally, the percentage of real memory being used by the process, this measure does not correlate with any other memory statistics
PGIN	Number of page ins caused by page faults. Since all I/O is classified as page faults, this is basically a measure of I/O volume
LIM	Always **xx**
TSIZ	Size of the text section of the executable file
TRS	Number of code-segment pages times 4
Environment	Value of all the environment variables for the process

Process priority alteration

Basically, if you have identified a process that is using too much CPU time, you can reduce its effective priority by increasing its nice value with the **renice** command.

For example:

```
renice +5 ProcID
```

The nice value of the *ProcID*'s would increase process from the normal 20 of a foreground process to 25. You must have root authority to reset the process *ProcID*'s nice value to 20. Type:

```
renice -5 ProcID
```

Process termination

Normally, you use the **kill** command to end a process.

The **kill** command sends a signal to the designated process. Depending on the type of signal and the nature of the program that is running in the process, the process might end or might keep running. The signals you send are:

Item	Description
SIGTERM	(signal 15) is a request to the program to terminate. If the program has a signal handler for SIGTERM that does not actually terminate the application, this **kill** may have no effect. This is the default signal sent by **kill**.
SIGKILL	(signal 9) is a directive to kill the process immediately. This signal cannot be caught or ignored.

It is typically better to issue SIGTERM rather than SIGKILL. If the program has a handler for SIGTERM, it can clean up and terminate in an orderly fashion. Type:

```
kill -term ProcessID
```

(The **-term** could be omitted.) If the process does not respond to the SIGTERM, type:

```
kill -kill ProcessID
```

You might notice occasional defunct processes, also called *zombies*, in your process table. These processes are no longer executing, have no system space allocated, but still retain their PID number. You can recognize a zombie process in the process table because it displays <defunct> in the CMD column. For example:

```
  UID   PID  PPID   C    STIME    TTY  TIME CMD
                              .
                              .
                              .
  lee 22392 20682   0   Jul 10      -  0:05 xclock
  lee 22536 21188   0   Jul 10  pts/0  0:00 /bin/ksh
  lee 22918 24334   0   Jul 10  pts/1  0:00 /bin/ksh
  lee 23526 22536  22                   0:00 <defunct>
  lee 24334 20682   0   Jul 10      ?  0:00 aixterm
  lee 24700     1   0   Jul 16      ?  0:00 aixterm
 root 25394 26792   2   Jul 16  pts/2  0:00 ksh
  lee 26070 24700   0   Jul 16  pts/3  0:00 /bin/ksh
  lee 26792 20082   0   Jul 10  pts/2  0:00 /bin/ksh
 root 27024 25394   2 17:10:44  pts/2  0:00 ps -ef
```

Zombie processes continue to exist in the process table until the parent process dies or the system is shut down and restarted. In the example shown above, the parent process (PPID) is the **ksh** command. When the Korn shell is exited, the defunct process is removed from the process table.

Sometimes a number of these defunct processes collect in your process table because an application has forked several child processes and has not exited. If this becomes a problem, the simplest solution is to modify the application so its **sigaction** subroutine ignores the **SIGCHLD** signal.

Related information

sigaction command

Binding or unbinding a process

You can bind a process to a processor or unbind a previously bound process.

You must have root user authority to bind or unbind a process you do not own.

On multiprocessor systems, you can bind a process to a processor or unbind a previously bound process from:

- SMIT

- command line

  Note: While binding a process to a processor might lead to improved performance for the bound process (by decreasing hardware-cache misses), overuse of this facility could cause individual processors to become overloaded while other processors are under used. The resulting bottlenecks could reduce overall throughput and performance. During normal operations, it is better to let the operating system assign processes to processors automatically, distributing system load across all processors. Bind only those processes that you know can benefit from being run on a single processor.

Binding or Unbinding a Process Tasks		
Task	***SMIT Fast Path***	***Command or File***
Binding a Process	**smit bindproc**	**bindprocessor -q**
Unbinding a Process	**smit ubindproc**	**bindprocessor -u**

Fixes for stalled or unwanted processes

Stalled or unwanted processes can cause problems with your terminal. Some problems produce messages on your screen that give information about possible causes.

To perform the following procedures, you must have either a second terminal, a modem, or a network login. If you do not have any of these, fix the terminal problem by rebooting your machine.

Choose the appropriate procedure for fixing your terminal problem:

Freeing a terminal taken over by processes
You can stop stalled or unwanted process.

Identify and stop stalled or unwanted processes by doing the following:

1. Determine the active processes running on the screen by typing the following **ps** command:

   ```
   ps -ef | pg
   ```

   The **ps** command shows the process status. The **-e** flag writes information about all processes (except kernel processes), and the **f** flag generates a full listing of processes including what the command name and parameters were when the process was created. The **pg** command limits output to a single page at a time, so information does not quickly scroll off the screen.

   Suspicious processes include system or user processes that use up excessive amounts of a system resource such as CPU or disk space. System processes such as **sendmail**, **routed**, and **lpd** frequently become runaways. Use the **ps -u** command to check CPU usage.
2. Determine who is running processes on this machine by using the **who** command:

   ```
   who
   ```

   The **who** command displays information about all users currently on this system, such as login name, workstation name, date, and time of login.
3. Determine if you need to stop, suspend, or change the priority of a user process.

   Note: You must have root authority to stop processes other than your own. If you terminate or change the priority of a user process, contact the process owner and explain what you have done.

   - Stop the process using the **kill** command. For example:

     ```
     kill 1883
     ```

     The **kill** command sends a signal to a running process. To stop a process, specify the process ID (PID), which is 1883 in this example. Use the **ps** command to determine the PID number of commands.
   - Suspend the process and run it in the background by using the ampersand (&). For example:

```
/u/bin1/prog1 &
```

The **&** signals that you want this process to run in the background. In a background process, the shell does not wait for the command to complete before returning the shell prompt. When a process requires more than a few seconds to complete, run the command in background by typing an **&** at the end of the command line. Jobs running in the background appear in the normal **ps** command.

- Change the priority of the processes that have taken over by using the following **renice** command:

```
renice 20 1883
```

The **renice** command alters the scheduling priority of one or more running processes. The higher the number, the lower the priority with 20 being the lowest priority.

In the previous example, **renice** reschedules process number 1883 to the lowest priority. It will run when there is a small amount of unused processor time available.

Responding to screen messages
Use this procedure to respond to and recover from screen messages.

1. Make sure the DISPLAY environment variable is set correctly. Use either of the following methods to check the DISPLAY environment:
   - Use the **setsenv** command to display the environment variables.

```
setsenv
```

The **setsenv** command displays the protected state environment when you logged in.

Determine if the DISPLAY variable has been set. In the following example, the DISPLAY variable does not appear, which indicates that the DISPLAY variable is not set to a specific value.

```
SYSENVIRON:
NAME=casey
TTY=/dev/pts/5
LOGNAME=casey
LOGIN=casey
```

OR

- Change the value of the DISPLAY variable. For example, to set it to the machine named `bastet` and terminal 0, enter:

```
DISPLAY=bastet:0
export DISPLAY
```

If not specifically set, the DISPLAY environment variable defaults to `unix:0` (the console). The value of the variable is in the format *name*:*number* where *name* is the host name of a particular machine, and *number* is the X server number on the named system.

2. Reset the terminal to its defaults using the following **stty** command:

```
stty sane
```

The **stty sane** command restores the “sanity” of the terminal drivers. The command outputs an appropriate terminal resetting code from the `/etc/termcap` file (or `/usr/share/lib/terminfo` if available).

3. If the Return key does not work correctly, reset it by typing:

```
^J stty sane ^J
```

The `^J` represents the Ctrl-J key sequence.

Running multiple queues using environment variables RT_MPC and RT_GRQ

The use of multiple queues increases the processor affinity of threads, but there is a special situation where you might want to counteract this effect.

When there is only one run queue, a thread that has been awakened (the waking thread) by another running thread (the waker thread) would normally be able to use the CPU immediately on which the waker thread was running. With multiple run queues, the waking thread may be on the run queue of another CPU which cannot notice the waking thread until the next scheduling decision. This may result in up to a 10 ms delay.

This is similar to scenarios in earlier releases of this operating system which might have occurred using the bindprocessor option. If all CPUs are constantly busy, and there are a number of interdependent threads waking up, there are two options available.

- The first option, which uses one run queue, is to set the environment variable `RT_GRQ=ON` which forces unbound selected threads to be dispatched off the global run queue.
- Alternatively, users can choose the real time kernel option (type the command `bosdebug -R on` and then `bosboot`) and the `RT_MPC=ON` environment variable for selected processes. It is essential to maintain a performance log of your systems to closely monitor the impact of any tuning you attempt.

System accounting

The system accounting utility allows you to collect and report on individual and group use of various system resources.

This accounting information can be used to bill users for the system resources they utilize, and to monitor selected aspects of the system operation. To assist with billing, the accounting system provides the resource-usage totals defined by members of the adm group, and, if the **chargefee** command is included, factors in the billing fee.

The accounting system also provides data to assess the adequacy of current resource assignments, set resource limits and quotas, forecast future needs, and order supplies for printers and other devices.

The following information should help you understand how to implement the accounting utility in your system.

Accounting data reports

After the various types of accounting data are collected, the records are processed and converted into reports.

Accounting commands automatically convert records into scientific notation when numbers become large. A number is represented in scientific notation in the following format:

*Base***e+***Exp*

*Base***e-***Exp*

which is the number equal to the *Base* number multiplied by 10 to the **+***Exp* or **-***Exp* power. For example, the scientific notation `1.345e+9` is equal to 1.345×10^{9}, or 1,345,000,000. And the scientific notation `1.345e-9` is equal to 1.345×10^{-9} or, 0.000000001345.

Related concepts

Process accounting data
The Accounting system collects data on resource usage for each process as it runs.

Daily accounting reports

To generate a daily report, use the **runacct** command.

This command summarizes data into an ASCII file named `/var/adm/acct/sum(x)/rprt`*MMDD*. *MMDD* specifies the month and day the report is run. The report covers the following:

- Daily report

- Daily Usage report
- Daily Command Summary
- Monthly Total Command Summary
- Last Login

Daily report

Daily accounting reports contain data on connect-time, processes, disk usage, printer usage, and fees to charge.

The **acctmerg** command merges raw accounting data on connect-time, processes, disk usage, printer usage, and fees to charge into daily reports. Called by the **runacct** command as part of its daily operation, the **acctmerg** command produces the following:

/var/adm/acct/nite(x)/dacct
An intermediate report that is produced when one of the input files is full.

/var/adm/acct/sum(x)/tacct
A cumulative total report in `tacct` format. This file is used by the **monacct** command to produce the ASCII monthly summary.

The **acctmerg** command can convert records between ASCII and binary formats and merge records from different sources into a single record for each user. For more information about the **acctmerg** command, see **acctmerg**.

The first line of the Daily report begins with the start and finish times for the data collected in the report, a list of system-level events including any existing shutdowns, reboots, and run-level changes. The total duration is also listed indicating the total number of minutes included within the accounting period (usually 1440 minutes, if the report is run every 24 hours). The report contains the following information:

Item	Description
LINE	Console, tty, or pty In use
MINUTES	Total number of minutes the line was in use
PERCENT	Percentage of time in the accounting period that the line was in use
# SESS	Number of new login sessions started
# ON	Same as **# SESS**
# OFF	Number of logouts plus interrupts made on the line

Daily Usage accounting report

The Daily Usage report is a summarized report of system usage per user ID during the accounting period.

Some fields are divided into prime and non-prime time, as defined by the accounting administrator in the `/usr/lib/acct/holidays` directory. The report contains the following information:

Item	Description
UID	User ID
LOGIN NAME	User name
CPU (PRIME/NPRIME)	Total CPU time for all of the user's processes in minutes
KCORE (PRIME/NPRIME)	Total memory used by running processes, in kilobyte-minutes
CONNECT (PRIME/NPRIME)	Total connect time (how long the user was logged in) in minutes
DISK BLOCKS	Average total amount of disk space used by the user on all filesystems for which accounting is enabled
FEES	Total fees entered with **chargefee** command
# OF PROCS	Total number of processes belonging to this user

Item	Description
# OF SESS	Number of distinct login sessions for this user
# DISK SAMPLES	Number of times disk samples were run during the accounting period. If no DISK BLOCKS are owned, the value will be zero

Daily Command Summary accounting report

The Daily Command Summary report shows each command executed during the accounting period, with one line per each unique command name.

The table is sorted by TOTAL KCOREMIN (described below), with the first line including the total information for all commands. The data listed for each command is cumulative for all executions of the command during the accounting period. The columns in this table include the following information:

Item	Description
COMMAND NAME	Command that was executed
NUMBER CMDS	Number of times the command executed
TOTAL KCOREMIN	Total memory used by running the command, in kilobyte-minutes
TOTAL CPU-MIN	Total CPU time used by the command in minutes
TOTAL REAL-MIN	Total real time elapsed for the command in minutes
MEAN SIZE-K	Mean size of memory used by the command per CPU minute
MEAN CPU-MIN	Mean numbr of CPU minutes per execution of the command
HOG FACTOR	Measurement of how much the command hogs the CPU while it is active. It is the ratio of **TOTAL CPU-MIN** over **TOTAL REAL-MIN**
CHARS TRNSFD	Number of characters transferred by the command with system reads and writes
BLOCKS READ	Number of physical block reads and writes performed by the command

Monthly Total Command Summary accounting report

The Monthly Total Command Summary, created by the **monacct** command, provides information about all commands executed since the previous monthly report.

The fields and information mean the same as those in the Daily Command Summary.

Last login

The Last Login report displays two fields for each user ID. The first field is **YY-MM-DD** and indicates the most recent login for the specified user. The second field is the name of the user account.

A date field of `00-00-00` indicates that the user ID has never logged in.

Accounting report summary

You can generate a report that summarizes raw accounting data.

To summarize raw accounting data, use the **sa** command. This command reads the raw accounting data, usually collected in the `/var/adm/pacct` file, and the current usage summary data in the `/var/adm/savacct` file, if summary data exists. It combines this information into a new usage summary report and purges the raw data file to make room for further data collection.

Prerequisites

The **sa** command requires an input file of raw accounting data such as the `pacct` file (process accounting file). To collect raw accounting data, you must have an accounting system set up and running.

Procedure

The purpose of the **sa** command is to summarize process accounting information and to display or store that information. The simplest use of the command displays a list of statistics about every process that has run during the life of the `pacct` file being read. To produce such a list, type:

```
/usr/sbin/sa
```

To summarize the accounting information and merge it into the summary file, type:

```
/usr/sbin/sa -s
```

The **sa** command offers many additional flags that specify how the accounting information is processed and displayed. See the **sa** command description for more information.

Related tasks

Setting up an accounting system
You can set up an accounting system.

Monthly report

You can generate a Monthly accounting report.

Called by the **cron** daemon, the **monacct** command produces the following:

Item	Description
`/var/adm/acct/fiscal`	A periodic summary report produced from the `/var/adm/acct/sum/tacct` report by the **monacct** command. The **monacct** command can be configured to run monthly or at the end of a fiscal period.

Connect-time reports

Accounting records include login, logout, system-shutdown, and lastlogin records.

The **runacct** command calls two commands, **acctcon1** and **acctcon2**, to process the login, logout, and system-shutdown records that collect in the `/var/adm/wtmp` file. The **acctcon1** command converts these records into session records and writes them to the `/var/adm/acct/nite(x)/lineuse` file. The `acctcon2` command then converts the session records into a total accounting record, `/var/adm/logacct`, that the **acctmerg** command adds to daily reports. For information about these commands, see **runacct**, **acctcon1**, and **acctcon2**.

If you run the **acctcon1** command from the command line, you must include the **-l** flag to produce the line-use report, `/var/adm/acct/nite(x)/lineuse`. To produce an overall session report for the accounting period, `/var/adm/acct/nite(x)/reboots`, use the **acctcon1** command with the **-o** flag.

The **lastlogin** command produces a report that gives the last date on which each user logged in. For information about the **lastlogin** command, see **lastlogin**.

Related concepts

Connect-time accounting data
Connect-time data is collected by the **init** command and the **login** command.

Disk-usage accounting data
Much accounting information is collected as the resources are consumed. The **dodisk** command, run as specified by the **cron** daemon, periodically writes disk-usage records for each user to the `/var/adm/acct/nite(x)/dacct` file.

Disk-usage accounting report

The disk-usage records collected in the `/var/adm/acct/nite(x)/dacct` file are merged into the daily accounting reports by the **acctmerg** command.

For information about the **acctmerg** command, see **acctmerg**.

Printer-Usage accounting report

The ASCII record in the `/var/adm/qacct` file can be converted to a total accounting record to be added to the daily report by the **acctmerg** command.

For information about the **acctmerg** command, see **acctmerg**.

Related concepts

Printer-usage accounting data
The collection of printer-usage data is a cooperative effort between the **enq** command and the queuing daemon.

Fee accounting report

If you used the **chargefee** command to charge users for services such as file restores, consulting, or materials, an ASCII total accounting record is written in the `/var/adm/fee` file. This file is added to the daily reports by the **acctmerg** command.

For information about the **chargefee** and **acctmerg** commands, see **chargefee** and **acctmerg**.

Related concepts

Fee accounting data
You can produce an ASCII total accounting record in the `/var/adm/fee` file.

Fiscal accounting reports

The Fiscal Accounting Reports generally collected monthly by using the **monacct** command.

The report is stored in `/var/adm/acct/fiscal(x)/fiscrptMM` where *MM* is the month that the **monacct** command was executed. This report includes information similar to the daily reports summarized for the entire month.

Accounting system activity reports

You can generate a report that shows Accounting system activity.

To generate a report on system activity, use the **prtacct** command. This command reads the information in a total accounting file (`tacct` file format) and produces formatted output. Total accounting files include the daily reports on connect time, process time, disk usage, and printer usage.

Prerequisites

The **prtacct** command requires an input file in the `tacct` file format. This implies that you have an accounting system set up and running or that you have run the accounting system in the past.

Procedure

Generate a report on system activity by entering:

```
prtacct -f Specification -v Heading File
```

Specification is a comma-separated list of field numbers or ranges used by the **acctmerg** command. The optional **-v** flag produces verbose output where floating-point numbers are displayed in higher precision notation. *Heading* is the title you want to appear on the report and is optional. *File* is the full path name of the total accounting file to use for input. You can specify more than one file.

Related tasks

Setting up an accounting system
You can set up an accounting system.

Greater than eight character username support

In order to maintain backwards compatibility with all scripts, long username support is not enabled by default within accounting. Instead, all user IDs are truncated to the first eight characters.

In order to enable long username support, most commands have been given the additional **-X** flag, which allows them to accept and output greater than eight-character user IDs (in both ASCII and binary

formats). In addition, when long username support is enabled, commands and scripts will process files in the /var/adm/acct/sumx, /var/adm/acct/nitex, and /var/adm/acct/fiscalx directories, instead of using /var/adm/acct/sum, /var/adm/acct/nite, and /var/adm/acct/fiscal.

Accounting commands

The accounting commands function several different ways.

Some commands:

- Collect data or produce reports for a specific type of accounting: connect-time, process, disk usage, printer usage, or command usage.
- Call other commands. For example, the **runacct** command, which is usually run automatically by the **cron** daemon, calls many of the commands that collect and process accounting data and prepare reports. To obtain automatic accounting, you must first configure the **cron** daemon to run the **runacct** command. See the **crontab** command for more information about how to configure the **cron** daemon to submit commands at regularly scheduled intervals. For information about these commands, see **runacct**, **cron** daemon, and **crontab**.
- Perform maintenance functions and ensure the integrity of active data files.
- Enable members of the adm group to perform occasional tasks, such as displaying specific records, by entering a command at the keyboard.
- Enable a user to display specific information. There is only one user command, the **acctcom** command, which displays process accounting summaries.

Commands that run automatically

Several commands automatically collect accounting data.

Several commands usually run by the **cron** daemon automatically collect accounting data. These commands are:

runacct
: Handles the main daily accounting procedure. Normally initiated by the **cron** daemon during non-prime hours, the **runacct** command calls several other accounting commands to process the active data files and produce command and resource usage summaries, sorted by user name. It also calls the **acctmerg** command to produce daily summary report files, and the **ckpacct** command to maintain the integrity of the active data files.

ckpacct
: Handles pacct file size. It is advantageous to have several smaller pacct files if you must restart the **runacct** procedure after a failure in processing these records. The **ckpacct** command checks the size of the /var/adm/pacct active data file, and if the file is larger than 500 blocks, the command invokes the **turnacct switch** command to turn off process accounting temporarily. The data is transferred to a new pacct file, /var/adm/pacct *x*. (*x* is an integer that increases each time a new pacct file is created.) If the number of free disk blocks falls below 500, the **ckpacct** command calls the **turnacct off** command to turn off process accounting.

dodisk
: Calls the **acctdisk** command and either the **diskusg** command or the **acctdusg** command to write disk-usage records to the /var/adm/acct/nite/dacct file. This data is later merged into the daily reports.

dodisk
: Calls the **acctdisk** command and either the **diskusg** command or the **acctdusg** command to write disk-usage records to the /var/adm/acct/nite/dacct file. This data is later merged into the daily reports.

monacct
: Produces a periodic summary from daily reports.

sa1
: Collects and stores binary data in the /var/adm/sa/sa *dd* file, where *dd* is the day of the month.

sa2
Writes a daily report in the/var/adm/sa/sa*dd* file, where *dd* is the day of the month. The command removes reports from the /var/adm/sa/sa*dd* file that have been there longer than one week.

Other commands are run automatically by procedures other than the **cron** daemon:

startup
When added to the /etc/rc file, the **startup** command initiates startup procedures for the accounting system.

shutacct
Records the time accounting was turned off by calling the **acctwtmp** command to write a line to /var/adm/wtmp file. It then calls the **turnacct off** command to turn off process accounting.

Keyboard commands
A member of the adm group can enter the following commands from the keyboard.

ac
Prints connect-time records. This command is provided for compatibility with Berkeley Software Distribution (BSD) systems.

acctcom
Displays process accounting summaries. This command is also available to users.

acctcon1
Displays connect-time summaries. Either the **-l** flag or the **-o** flag must be used.

accton
Turns process accounting on and off.

chargefee
Charges the user a predetermined fee for units of work performed. The charges are added to the daily report by the **acctmerg** command.

fwtmp
Converts files between binary and ASCII formats.

last
Displays information about previous logins. This command is provided for compatibility with BSD systems.

lastcomm
Displays information about the last commands that were executed. This command is provided for compatibility with BSD systems.

lastlogin
Displays the time each user last logged in.

pac
Prepares printer/plotter accounting records. This command is provided for compatibility with BSD systems.

prctmp
Displays a session record.

prtacct
Displays total accounting files.

sa
Summarizes raw accounting information to help manage large volumes of accounting information. This command is provided for compatibility with BSD systems.

sadc
Reports on various local system actions, such as buffer usage, disk and tape I/O activity, TTY device activity counters, and file access counters.

sar
Writes to standard output the contents of selected cumulative activity counters in the operating system. The **sar** command reports only on local activities.

time
Prints real time, user time, and system time required to run a command.

timex
Reports in seconds the elapsed time, user time, and run time.

Related concepts

System data collection and reporting
You can set up the system to automatically collect data and generate reports.

Accounting files

The two main accounting directories are the `/usr/sbin/acct` directory, where all the C language programs and shell procedures needed to run the accounting system are stored, and the `/var/adm` directory, which contains the data, report and summary files.

The accounting data files belong to members of the adm group, and all active data files (such as `wtmp` and `pacct`) reside in the adm home directory `/var/adm`.

Accounting data files

The following files are in the `/var/adm` directory.

Item	Description
`/var/adm/diskdiag`	Diagnostic output during the running of disk accounting programs
`/var/adm/dtmp`	Output from the **acctdusg** command
`/var/adm/fee`	Output from the **chargefee** command, in ASCII `tacct` records
`/var/adm/pacct`	Active process accounting file
`/var/adm/wtmp`	Active process accounting file
`/var/adm/Spacct`*.mmdd*	Process accounting files for *mmdd* during the execution of the **runacct** command.

Accounting report and summary files

Some subdirectories are needed before you enable the Accounting system.

Report and summary files reside in a `/var/adm/acct` subdirectory. You must create the following subdirectories before the Accounting system is enabled.

/var/adm/acct/nite(x)
Contains files that the **runacct** command reuses daily

/var/adm/acct/sum(x)
Contains the cumulative summary files that the **runacct** command updates daily

/var/adm/acct/fiscal(x)
Contains the monthly summary files that the **monacct** command creates.

Related tasks

Setting up an accounting system
You can set up an accounting system.

Starting the runacct command for accounting

You can start the **runacct** command.

Prerequisites

1. You must have the accounting system installed.
2. You must have root user or adm group authority.

Notes:

1. If you call the **runacct** command with no parameters, the command assumes that this is the first time that the command has been run today. Therefore, you need to include the *mmdd* parameter when you restart the **runacct** program, so that the month and day are correct. If you do not specify a state, the **runacct** program reads the `/var/adm/acct/nite(x)/statefile` file to determine the entry point for processing. To override the `/var/adm/acct/nite(x)/statefile` file, specify the desired state on the command line.
2. When you perform the following task, you might need to use the full path name `/usr/sbin/acct/runacct` rather than the simple command name, **runacct**.

Procedure

To start the **runacct** command, type the following:

```
nohup runacct 2> \
/var/adm/acct/nite/accterr &
```

This entry causes the command to ignore all **INTR** and **QUIT** signals while it performs background processing. It redirects all standard error output to the `/var/adm/acct/nite/accterr` file.

Restarting the runacct command for Accounting

If the **runacct** command is unsuccessful, you can restart it.

The prerequisites for this procedure are:

- You must have the accounting system installed.
- You must have root user or adm group authority.

Note: The most common reason why the **runacct** command can fail are because:

- The system goes down.
- The `/usr` file system runs out of space.
- The `/var/adm/wtmp` file has records with inconsistent date stamps.

If the **runacct** command is unsuccessful, do the following:

1. Check the `/var/adm/acct/nite(x)/active` *mmdd* file for error messages.
2. If both the active file and lock files exist in `acct/nite`, check the `accterr` file, where error messages are redirected when the **cron** daemon calls the **runacct** command.
3. Perform any actions needed to eliminate errors.
4. Restart the **runacct** command.
5. To restart the **runacct** command for a specific date, type the following:

   ```
   nohup runacct 0601 2>> \
   /var/adm/acct/nite/accterr &
   ```

   This restarts the **runacct** program for June 1 (`0601`). The **runacct** program reads the `/var/adm/acct/nite/statefile` file to find out with which state to begin. All standard error output is appended to the `/var/adm/acct/nite/accterr` file.
6. To restart the **runacct** program at a specified state, for example, the MERGE state, type the following:

   ```
   nohup runacct 0601 MERGE 2>> \
   /var/adm/acct/nite/accterr &
   ```

runacct command files

The **runacct** command produces report and summary files.

The following report and summary files, produced by the **runacct** command, are of particular interest:

Item	Description
/var/adm/acct/nite(x)/lineuse	Contains usage statistics for each terminal line on the system. This report is especially useful for detecting bad lines. If the ratio between the number of logouts and logins exceeds about 3 to 1, there is a good possibility that a line is failing.
/var/adm/acct/nite(x)/daytacct	Contains the total accounting file for the previous day.
/var/adm/acct/sum(x)/tacct	Contains the accumulation of each day's nite/daytacct file and can be used for billing purposes. The **monacct** command restarts the file each month or fiscal period.
/var/adm/acct/sum(x)/cms	Contains the accumulation of each day's command summaries. The **monacct** command reads this binary version of the file and purges it. The ASCII version is nite/cms.
/var/adm/acct/sum(x)/daycms	Contains the daily command summary. An ASCII version is stored in nite/daycms.
/var/adm/acct/sum(x)/loginlog	Contains a record of the last time each user ID was used.
/var/adm/acct/sum(x)/rprt *mmdd*	This file contains a copy of the daily report saved by the **runacct** command.

Files in the /var/adm/acct/nite(x) directory

The following files are in the /var/adm/acct/nite(x) directory.

Item	Description
active	Used by the **runacct** command to record progress and print warning and error messages. The file active. *mmdd* is a copy of the active file made by the **runacct** program after it detects an error.
cms	ASCII total command summary used by the **prdaily** command.
ctacct.*mmdd*	Connect total accounting records.
ctmp	Connect session records.
daycms	ASCII daily command summary used by the **prdaily** command.
daytacct	Total accounting records for one day.
dacct	Disk total accounting records, created by the **dodisk** command.
accterr	Diagnostic output produced during the execution of the **runacct** command.
lastdate	Last day the **runacct** executed, in *date +%m%d* format.
lock1	Used to control serial use of the **runacct** command.
lineuse	tty line usage report used by the **prdaily** command.
log	Diagnostic output from the **acctcon1** command.
log*mmdd*	Same as **log** after the **runacct** command detects an error.
reboots	Contains beginning and ending dates from wtmp, and a listing of system restarts.
statefile	Used to record the current state during execution of the **runacct** command.
tmpwtmp	wtmp file corrected by the **wtmpfix** command.

Item	Description
wtmperror	Contains **wtmpfix** error messages.
wtmperr*mmdd*	Same as wtmperror after the **runacct** command detects an error.
wtmp.*mmdd*	Contains previous day's wtmp file. Removed during the cleanup of **runacct** command.

Files in the /var/adm/acct/sum(x) directory

The following files are in the /var/adm/acct/sum(x) directory.

Item	Description
cms	Total command summary file for the current fiscal period, in binary format.
cmsprev	Command summary file without the latest update.
daycms	Command summary file for the previous day, in binary format.
lastlogin	File created by the **lastlogin** command.
pacct.*mmdd*	Concatenated version of all pacct files for *mmdd*. This file is removed after system startup by the **remove** command. For information about the **remove** command, see **remove**.
rprt*mmdd*	Saved output of the **prdaily** command.
tacct	Cumulative total accounting file for the current fiscal period.
tacctprev	Same as tacct without the latest update.
tacct*mmdd*	Total accounting file for *mmdd*.

Files in the /var/adm/acct/fiscal(x) directory

The following files are in the /var/adm/acct/fiscal(x) directory.

Item	Description
cms*?*	Total command summary file for the fiscal period, specified by *?*, in binary format
fiscrpt*?*	A report similar to that of the **prdaily** command for fiscal period, specified by *?*, in binary format
tacct*?*	Total accounting file for fiscal period, specified by *?*, in binary format.

Accounting file formats

The following table summarizes the accounting file output and formats.

Item	Description
wtmp	Produces the active process accounting file. The format of the wtmp file is defined in the utmp.h file. For information about the utmp.h file, see utmp.h.
ctmp	Produces connect session records. The format is described in the ctmp.h file.
pacct*	Produces active process accounting records. The format of the output is defined in the /usr/include/sys/acct.h file.
Spacct*	Produces process accounting files for *mmdd* during the running of the **runacct** command. The format of these files is defined in the sys/acct.h file.
daytacct	Produces total accounting records for one day. The format of the file is defined in the tacct file format.
sum/tacct	Produces binary file that accumulates each day's command summaries. The format of this file is defined in the /usr/include/sys/acct.h header file.

Item	Description
ptacct	Produces concatenated versions of pacct files. The format of these files are defined in the tacct file.
ctacct	Produces connect total accounting records. The output of this file is defined in the tacct file.
cms	Produces total accounting command summary used by the **prdaily** command, in binary format. The ASCII version is nite/cms.
daycms	Daily command summary used by the **prdaily** command, in binary format. The ASCII version is nite/daycms.

Administering system accounting

There are multiple tasks you can complete for system accounting. These tasks include setting up an accounting system, showing CPU usage, and displaying accounting processes.

Setting up an accounting system

You can set up an accounting system.

You must have root authority to complete this procedure.

The information below is an overview of the steps you must take to set up an accounting system. Refer to the commands and files noted in these steps for more specific information.

1. Use the **nulladm** command to ensure that each file has the correct access permission: read (r) and write (w) permission for the file owner and group and read (r) permission for others by typing:

   ```
   /usr/sbin/acct/nulladm wtmp pacct
   ```

   This provides access to the pacct and wtmp files.
2. Update the /etc/acct/holidays file to include the hours you designate as prime time and to reflect your holiday schedule for the year.

   Note: Comment lines can appear anywhere in the file as long as the first character in the line is an asterisk (*).

   a. To define prime time, fill in the fields on the first data line (the first line that is not a comment), using a 24-hour clock. This line consists of three 4-digit fields, in the following order:

      i) Current year

      ii) Beginning of prime time (*hhmm*)

      iii) End of prime time (*hhmm*)

      Leading blanks are ignored. You can enter midnight as either 0000 or 2400.

      For example, to specify the year 2000, with prime time beginning at 8:00 a.m. and ending at 5:00 p.m., enter:

      ```
      2000  0800  1700
      ```

   b. To define the company holidays for the yea, fill in the next data line. Each line contains four fields, in the following order:

      i) Day of the year

      ii) Month

      iii) Day of the month

      iv) Description of holiday

The day-of-the-year field contains the number of the day on which the holiday falls and must be a number from 1 through 365 (366 on leap year). For example, February 1st is day 32. The other three fields are for information only and are treated as comments.

A two-line example follows:

```
  1  Jan  1  New Year's Day
332  Nov 28  Thanksgiving Day
```

3. Turn on process accounting by adding the following line to the /etc/rc file or by deleting the comment symbol (#) in front of the line if it exists:

```
/usr/bin/su - adm -c /usr/sbin/acct/startup
```

The **startup** procedure records the time that accounting was turned on and cleans up the previous day's accounting files.

4. Identify each file system you want included in disk accounting by adding the following line to the stanza for the file system in the /etc/filesystems file:

```
account = true
```

5. Specify the data file to use for printer data by adding the following line to the queue stanza in the /etc/qconfig file:

```
acctfile = /var/adm/qacct
```

6. As the adm user, create a /var/adm/acct/nite, a /var/adm/acct/fiscal, a and /var/adm/acct/sum directory to collect daily and fiscal period records:

```
su - adm
cd /var/adm/acct
mkdir nite fiscal sum
exit
```

For long usernames, use the following commands instead:

```
su - adm
cd /var/adm/acct
mkdir nitex fiscalx sumx
exit
```

7. Set daily accounting procedures to run automatically by editing the /var/spool/cron/crontabs/adm file to include the **dodisk**, **ckpacct**, and **runacct** commands.
For example:

```
0 2 * * 4 /usr/sbin/acct/dodisk
5 * * * * /usr/sbin/acct/ckpacct
0 4 * * 1-6 /usr/sbin/acct/runacct
             2>/var/adm/acct/nite/accterr
```

For long usernames, add the following lines instead:

```
0 2 * * 4 /usr/sbin/acct/dodisk -X
5 * * * * /usr/sbin/acct/ckpacct
0 4 * * 1-6 /usr/sbin/acct/runacct -X
             2>/var/adm/acct/nitex/accterr
```

The first line starts disk accounting at 2:00 a.m. (0 2) each Thursday (4). The second line starts a check of the integrity of the active data files at 5 minutes past each hour (5 *) every day (*). The third line runs most accounting procedures and processes active data files at 4:00 a.m. (0 4) every Monday through Saturday (1-6). If these times do not fit the hours your system operates, adjust your entries.

Note: You must have root user authority to edit the /var/spool/cron/crontabs/adm file.

8. Set the monthly accounting summary to run automatically by including the **monacct** command in the /var/spool/cron/crontabs/adm file.
For example, type:

```
15 5 1 * * /usr/sbin/acct/monacct
```

For long usernames, add the following line instead:

```
15 5 1 * * /usr/sbin/acct/monacct -X
```

Be sure to schedule this procedure early enough to finish the report. This example starts the procedure at 5:15 a.m. on the first day of each month.

9. To submit the edited `cron` file, type:

```
crontab /var/spool/cron/crontabs/adm
```

Related concepts

System data collection and reporting
You can set up the system to automatically collect data and generate reports.

Accounting system activity reports
You can generate a report that shows Accounting system activity.

Accounting report summary
You can generate a report that summarizes raw accounting data.

Related tasks

Displaying the process time of active Accounting processes
You can display the process time for active processes.

Displaying the process time of finished Accounting processes
You can display the process time of finished processes.

Showing the CPU usage for each accounting process
You can display formatted reports about the CPU usage by user with the **acctprc1** command.

Showing the CPU accounting usage for each user
You can display a formatted report about the CPU usage by user with a combination of the **acctprc1** and **prtacct** commands.

Displaying printer or plotter usage accounting records
You can display printer or plotter usage accounting records with the **pac** command.

Related reference

Accounting report and summary files
Some subdirectories are needed before you enable the Accounting system.

Displaying Accounting system activity

You can display formatted information about system activity with the **sar** command.

To display system activity statistics, the **sadc** command must be running.

Note: The typical method of running the **sadc** command is to place an entry for the **sa1** command in the root `crontab` file. The **sa1** command is a shell-procedure variant of the **sadc** command designed to work with the **cron** daemon.

To display basic system-activity information, type:

```
sar 2 6
```

where the first number is the number of seconds between sampling intervals and the second number is the number of intervals to display. The output of this command looks something like this:

```
    arthurd 2 3 000166021000    05/28/92
```

```
14:03:40    %usr    %sys    %wio   %idle
14:03:42       4       9       0      88
14:03:43       1      10       0      89
14:03:44       1      11       0      88
```

```
14:03:45       1      11       0      88
14:03:46       3       9       0      88
14:03:47       2      10       0      88

Average        2      10       0      88
```

The **sar** command also offers a number of flags for displaying an extensive array of system statistics. To see all available statistics, use the **-A** flag. For a list of the available statistics and the flags for displaying them, see the **sar** command.

Note: To have a daily system activity report written to `/var/adm/sa/sa`*dd*, include an entry in the root `crontab` file for the **sa2** command. The **sa2** command is a shell procedure variant for the **sar** command designed to work with the **cron** daemon.

Showing Accounting system activity while running a command

You can display formatted information about system activity while a particular command is running.

The **-o** and **-p** flags of the **timex** command require that system accounting be turned on.

You can use the **time** and **timex** commands to display formatted information about system activity while a particular command is running.

To display the elapsed time, user time, and system execution time for a particular command, type:

```
time CommandName
```

OR

```
timex CommandName
```

To display the total system activity (all the data items reported by the **sar** command) during the execution of a particular command, type:

```
timex -s CommandName
```

The **timex** command has two additional flags. The **-o** flag reports the total number of blocks read or written by the command and all of its children. The **-p** flag lists all of the process accounting records for a command and all of its children.

Displaying the process time of active Accounting processes

You can display the process time for active processes.

The **acctcom** command reads input in the total accounting record form (`acct` file format). This implies that you have process accounting turned on or that you have run process accounting in the past.

The **ps** command offers a number of flags to tailor the information displayed.

To produce a full list of all active processes except kernel processes, type:

```
ps -ef
```

You can also display a list of all processes associated with terminals. To do this, type:

```
ps -al
```

Both of these usages display a number of columns for each process, including the current CPU time for the process in minutes and seconds.

Related tasks

Setting up an accounting system

You can set up an accounting system.

Displaying the process time of finished Accounting processes

You can display the process time of finished processes.

The **acctcom** command reads input in the total accounting record form (`acct` file format). This implies that you have process accounting turned on or that you have run process accounting in the past.

The process accounting functions are turned on with the **startup** command, which is typically started at system initialization with a call in the `/etc/rc` file. When the process accounting functions are running, a record is written to `/var/adm/pacct` (a total accounting record file) for every finished process that includes the start and stop time for the process. You can display the process time information from a `pacct` file with the **acctcom** command. This command has a number of flags that allow flexibility in specifying which processes to display.

For example, to see all processes that ran for a minimum number of CPU seconds or longer, use the **-O** flag, type:

```
acctcom -O 2
```

This displays records for every process that ran for at least 2 seconds. If you do not specify an input file, the **acctcom** command reads input from the `/var/adm/pacct` directory.

Related tasks

Setting up an accounting system
You can set up an accounting system.

Showing the CPU usage for each accounting process

You can display formatted reports about the CPU usage by user with the **acctprc1** command.

The **acctprc1** command requires input in the total accounting record form (`acct` file format). This implies that you have process accounting turned on or that you have run process accounting in the past.

To produce a formatted report of CPU usage by process, type:

```
acctprc1 </var/adm/pacct
```

Related tasks

Setting up an accounting system
You can set up an accounting system.

Showing the CPU accounting usage for each user

You can display a formatted report about the CPU usage by user with a combination of the **acctprc1** and **prtacct** commands.

The acctprc1, acctprc2, or accton Command command requires input in the total accounting record form (`acct` file format). This implies that you have process accounting turned on or that you have run process accounting in the past.

To show the CPU usage for each user, perform the following steps:

1. Produce an output file of CPU usage by process by typing:

   ```
   acctprc1 </var/adm/pacct >out.file
   ```

   The `/var/adm/pacct` file is the default output for process accounting records. You might want to specify an archive `pacct` file instead.
2. Produce a binary total accounting record file from the output of the previous step by typing:

   ```
   acctprc2 <out.file >/var/adm/acct/nite/daytacct
   ```

   Note: The `daytacct` file is merged with other total accounting records by the **acctmerg** command to produce the daily summary record, `/var/adm/acct/sum(x)/tacct`.

3. Use the Showing the CPU accounting usage for each user command to display a formatted report of CPU usage summarized by the user by typing:

```
prtacct </var/adm/acct/nite/daytacct
```

Related tasks

Setting up an accounting system
You can set up an accounting system.

Displaying connect time usage for accounting

You can display the connect time of all users, of individual users, and by individual login with the **ac** command.

The **ac** command extracts login information from the `/var/adm/wtmp` file, so this file must exist. If the file has not been created, the following error message is returned:

```
No /var/adm/wtmp
```

If the file becomes too full, additional `wtmp` files are created; you can display connect-time information from these files by specifying them with the **-w** flag. For more information about the **ac** command, see **ac**.

To display the total connect time for all users, type:

```
/usr/sbin/acct/ac
```

This command displays a single decimal number that is the sum total connect time, in minutes, for all users who have logged in during the life of the current `wtmp` file.

To display the total connect time for one or more particular users, type:

```
/usr/sbin/acct/ac User1 User2 ...
```

This command displays a single decimal number that is the sum total connect time, in minutes, for the user or users you specified for any logins during the life of the current `wtmp` file.

To display the connect time by individual user plus the total connect time, type:

```
/usr/sbin/acct/ac -p User1 User2 ...
```

This command displays as a decimal number for each user specified equal to the total connect time, in minutes, for that user during the life of the current `wtmp` file. It also displays a decimal number that is the sum total connect time for all the users specified. If no user is specified in the command, the list includes all users who have logged in during the life of the `wtmp` file.

Displaying disk space utilization for accounting

You can display disk space utilization information with the **acctmerg** command.

To display disk space utilization information, the **acctmerg** command requires input from a `dacct` file (disk accounting). The collection of disk-usage accounting records is performed by the **dodisk** command.

To display disk space utilization information, type:

```
acctmerg -a1 -2,13 -h </var/adm/acct/nite(x)/dacct
```

This command displays disk accounting records, which include the number of 1 KB blocks utilized by each user.

Note: The **acctmerg** command always reads from standard input and can read up to nine additional files. If you are not piping input to the command, you must redirect input from one file; the rest of the files can be specified without redirection.

Displaying printer or plotter usage accounting records

You can display printer or plotter usage accounting records with the **pac** command.

- To collect printer usage information, you must have an accounting system set up and running. See "Setting up an accounting system" on page 161 for guidelines.
- The printer or plotter for which you want accounting records must have an `acctfile=` clause in the printer stanza of the `/etc/qconfig` file. The file specified in the `acctfile=` clause must grant read and write permissions to the root user or printq group.
- If the **-s** flag of the **pac** command is specified, the command rewrites the summary file name by appending **_sum** to the path name specified by the `acctfile=` clause in the `/etc/qconfig` file. This file must exist and grant read and write permissions to the root user or printq group.

To display printer usage information for all users of a particular printer, type:

```
/usr/sbin/pac -PPrinter
```

If you do not specify a printer, the default printer is named by the PRINTER environment variable. If the PRINTER variable is not defined, the default is `lp0`.

To display printer usage information for particular users of a particular printer, type:

```
/usr/sbin/pac -PPrinter User1 User2 ...
```

The **pac** command offers other flags for controlling what information gets displayed.

Related tasks

Setting up an accounting system
You can set up an accounting system.

Updating the holidays file

The Holidays file is out of date after the last holiday listed has passed or the year has changed. You can update the Holidays file.

The **acctcon1** command (started from the **runacct** command) sends mail to the **root** and **adm** accounts when the `/usr/lib/acct/holidays` file gets out of date.

Update the out-of-date Holidays file by editing the `/var/adm/acct/holidays` file to differentiate between prime and nonprime time.

Prime time is assumed to be the period when your system is most active, such as workdays. Saturdays and Sundays are always nonprime times for the accounting system, as are any holidays that you list.

The holidays file contains three types of entries: comments, the year and prime-time period, and a list of holidays as in the following example:

```
* Prime/Non-Prime Time Table for Accounting System
*
*   Curr       Prime             Non-Prime
*   Year       Start             Start
    1992       0830              1700
*
*   Day of     Calendar          Company
*   Year       Date              Holiday
*
*   1          Jan 1             New Year's Day
*   20         Jan 20            Martin Luther King Day
*   46         Feb 15            President's Day
*   143        May 28            Memorial Day
*   186        Jul 3             4th of July
*   248        Sep 7             Labor Day
*   329        Nov 24            Thanksgiving
*   330        Nov 25            Friday after
*   359        Dec 24            Christmas Eve
*   360        Dec 25            Christmas Day
*   361        Dec 26            Day after Christmas
```

The first noncomment line must specify the current year (as four digits) and the beginning and end of prime time, also as four digits each. The concept of prime and nonprime time only affects the way that the accounting programs process the accounting records.

If the list of holidays is too long, the **acctcon1** command generates an error, and you will need to shorten your list. You are safe with 20 or fewer holidays. If you want to add more holidays, just edit the holidays file each month.

Collecting accounting data

Once you have setup system accounting you will want to start collecting and processing the different type of accounting data.

System data collection and reporting

You can set up the system to automatically collect data and generate reports.

For data to be collected automatically, a member of the adm group must have been setup as an accounting system. The accounting system setup enables the **cron** daemon to run the commands that generate data on:

- The amount of time each user spends logged in to the system
- Usage of the processing unit, memory, and I/O resources
- The amount of disk space occupied by each user's files
- Usage of printers and plotters
- The number of times a specific command is given.

The system writes a record of each session and process after they are completed. These records are converted into total accounting (`tacct`) records arranged by user and merged into a daily report. Periodically, the daily reports are combined to produce totals for the defined fiscal period. Methods for collecting and reporting the data and the various accounting commands and files are discussed in the following sections.

Although most of the accounting data is collected and processed automatically, a member of the adm group can enter certain commands from the keyboard to obtain specific information.

Related tasks

Setting up an accounting system
You can set up an accounting system.

Related reference

Keyboard commands
A member of the adm group can enter the following commands from the keyboard.

Connect-time accounting data

Connect-time data is collected by the **init** command and the **login** command.

When you log in, the **login** program writes a record in the `/etc/utmp` file. This record includes your user name, the date and time of the login, and the login port. Commands, such as **who**, use this file to find out which users are logged into the various display stations. If the `/var/adm/wtmp` connect-time accounting file exists, the **login** command adds a copy of this login record to it. For information about the **init** and **login** commands, see **init** and **login**.

When your login program ends (normally when you log out), the **init** command records the end of the session by writing another record in the `/var/adm/wtmp` file. Logout records differ from login records in that they have a blank user name. Both the login and logout records have the form described in the `utmp.h` file. For information about the `utmp.h` file, see `utmp.h`.

The **acctwtmp** command also writes special entries in the `/var/adm/wtmp` file concerning system shutdowns and startups.

Related concepts

Connect-time reports
Accounting records include login, logout, system-shutdown, and lastlogin records.

Process accounting data

The Accounting system collects data on resource usage for each process as it runs.

This data includes:

- The user and group numbers under which the process runs
- The first eight characters of the name of the command
- A 64-bit numeric key representing the Workload Manager class that the process belongs to
- The elapsed time and processor time used by the process
- Memory use
- The number of characters transferred
- The number of disk blocks read or written on behalf of the process

The **accton** command records these data in a specified file, usually the `/var/adm/pacct` file. For more information about the **accton** command, see **accton**.

Related commands are the **startup** command, the **shutacct** command, the **dodisk** command, the **ckpacct** command, and the **turnacct** command. For information about these commands, see **startup**, **shutacct**, **dodisk**, **ckpacct**, and **turnacct**.

Related concepts

Accounting data reports
After the various types of accounting data are collected, the records are processed and converted into reports.

Process accounting reports

Two commands process the billing-related data that was collected in the `/var/adm/pacct` or other specified file.

The **acctprc1** command translates the user ID into a user name and writes ASCII records containing the chargeable items (prime and non-prime CPU time, mean memory size, and I/O data). The **acctprc2** command transforms these records into total accounting records that are added to daily reports by the **acctmerg** command. For more information about the **acctmerg** command, see **acctmerg**.

Process accounting data also provides information that you can use to monitor system resource usage. The **acctcms** command summarizes resource use by command name. This provides information on how many times each command was run, how much processor time and memory was used, and how intensely the resources were used (also known as the *hog factor*). The **acctcms** command produces long-term statistics on system utilization, providing information on total system usage and the frequency with which commands are used. For more information about the **acctcms** command, see **acctcms**.

The **acctcom** command handles the same data as the **acctcms** command, but provides detailed information about each process. You can display all process accounting records or select records of particular interest. Selection criteria include the load imposed by the process, the time period when the process ended, the name of the command, the user or group that invoked the process, the name of the WLM class the proces belonged to, and the port at which the process ran. Unlike other accounting commands, **acctcom** can be run by all users. For more information about the **acctcom** command, see **acctcom**.

Disk-usage accounting data

Much accounting information is collected as the resources are consumed. The **dodisk** command, run as specified by the **cron** daemon, periodically writes disk-usage records for each user to the `/var/adm/acct/nite(x)/dacct` file.

To accomplish this, the **dodisk** command calls other commands. Depending upon the thoroughness of the accounting search, the **diskusg** command or the **acctdusg** command can be used to collect data.

The **acctdisk** command is used to write a total accounting record. The total accounting record, in turn, is used by the **acctmerg** command to prepare the daily accounting report.

The **dodisk** command charges a user for the links to files found in the user's login directory and evenly divides the charge for each file between the links. This distributes the cost of using a file over all who use it and removes the charges from users when they relinquish access to a file. For more information about the **dodisk** command and **cron** daemon, see **dodisk** and **cron**.

Related concepts

Connect-time reports
Accounting records include login, logout, system-shutdown, and lastlogin records.

Printer-usage accounting data

The collection of printer-usage data is a cooperative effort between the **enq** command and the queuing daemon.

The **enq** command enqueues the user name, job number, and the name of the file to be printed. After the file is printed, the **qdaemon** command writes an ASCII record to a file, usually the `/var/adm/qacct` file, containing the user name, user number, and the number of pages printed. You can sort these records and convert them to total accounting records. For more information about these commands, see **enq** and **qdaemon**.

Related concepts

Printer-Usage accounting report
The ASCII record in the `/var/adm/qacct` file can be converted to a total accounting record to be added to the daily report by the **acctmerg** command.

Fee accounting data

You can produce an ASCII total accounting record in the `/var/adm/fee` file.

You can enter the **chargefee** command to produce an ASCII total accounting record in the `/var/adm/fee` file. This file will be added to daily reports by the **acctmerg** command.

For information about the **chargefee** and **acctmerg** commands, see **chargefee** and **acctmerg**.

Related concepts

Fee accounting report
If you used the **chargefee** command to charge users for services such as file restores, consulting, or materials, an ASCII total accounting record is written in the `/var/adm/fee` file. This file is added to the daily reports by the **acctmerg** command.

Troubleshooting system accounting

Use these troubleshooting methods to tackle some of the basic problems that may occur when using system accounting. If the troubleshooting information does not address your problem, contact your service representative.

Fixing tacct errors

If you are using the accounting system to charge users for system resources, the integrity of the `/var/adm/acct/sum/tacct` file is quite important. Occasionally, mysterious **tacct** records appear that contain negative numbers, duplicate user numbers, or a user number of 65,535. You can fix these problems.

You must have root user or adm group authority.

To patch a tacct file, perform the following steps:

1. Move to the `/var/adm/acct/sum` directory by typing:

   ```
   cd /var/adm/acct/sum
   ```

2. Use the **prtacct** command to check the total accounting file, `tacctprev`, by typing:

```
prtacct tacctprev
```

The **prtacct** command formats and displays the tacctprev file so that you can check connect time, process time, disk usage, and printer usage.

3. If the tacctprev file looks correct, change the latest tacct .*mmdd* file from a binary file to an ASCII file. In the following example, the **acctmerg** command converts the tacct.*mmdd* file to an ASCII file named tacct.new:

```
acctmerg -v < tacct.mmdd > tacct.new
```

Note: The **acctmerg** command with the **-a** flag also produces ASCII output. The **-v** flag produces more precise notation for floating-point numbers.

The **acctmerg** command is used to merge the intermediate accounting record reports into a cumulative total report (**tacct**). This cumulative total is the source from which the **monacct** command produces the ASCII monthly summary report. Since the **monacct** command procedure removes all the tacct.*mmdd* files, you recreate the tacct file by merging these files.

4. Edit the tacct.new file to remove the bad records and write duplicate user number records to another file by typing:

```
acctmerg -i < tacct.new > tacct.mmdd
```

5. Create the tacct file again by typing:

```
acctmerg tacctprev < tacct.mmdd > tacct
```

Fixing wtmp errors

The /var/adm/wtmp, or "who temp" file, might cause problems in the day-to-day operation of the accounting system. You can fix wtmp errors.

You must have root user or adm group authority to perform this procedure.

When the date is changed and the system is in multiuser mode, date change records are written to the /var/adm/wtmp file. When a date change is encountered, the **wtmpfix** command adjusts the time stamps in the wtmp records. Some combinations of date changes and system restarts may slip past the **wtmpfix** command and cause the **acctcon1** command to fail and the **runacct** command to send mail to the **root** and **adm** accounts listing incorrect dates.

To fix wtmp errors, perform the following procedure:

1. Move to the /var/adm/acct/nite directory by typing:

```
cd /var/adm/acct/nite
```

2. Convert the binary wtmp file to an ASCII file that you can edit by typing:

```
fwtmp < wtmp.mmdd > wtmp.new
```

The **fwtmp** command converts wtmp from binary to ASCII.

3. Edit the ASCII wtmp.new file to delete damaged records or all records from the beginning of the file up to the needed date change by typing:

```
vi wtmp.new
```

4. Convert the ASCII wtmp.new file back to binary format by typing:

```
fwtmp -ic < wtmp.new > wtmp.mmdd
```

5. If the wtmp file is beyond repair, use the **nulladm** command to create an empty wtmp file.

This prevents any charges in the connect time.

```
nulladm wtmp
```

The **nulladm** command creates the file specified with read and write permissions for the file owner and group, and read permissions for other users. It ensures that the file owner and group are **adm**.

Related tasks

Fixing Accounting errors
You can correct date and time-stamp inconsistencies.

Fixing incorrect Accounting file permissions

To use the Accounting system, file ownership and permissions must be correct.

You must have root user or adm group authority to perform this procedure.

The **adm** administrative account owns the accounting command and scripts, except for `/var/adm/acct/accton` which is owned by root.

To fix incorrect Accounting file permissions, perform the following procedure:

1. To check file permissions using the **ls** command, type:

```
ls -l /var/adm/acct

-rws--x--- 1 adm adm 14628 Mar 19 08:11 /var/adm/acct/fiscal
-rws--x--- 1 adm adm 14628 Mar 19 08:11 /var/adm/acct/nite
-rws--x--- 1 adm adm 14628 Mar 19 08:11 /var/adm/acct/sum
```

2. Adjust file permissions with the **chown** command, if necessary.

   The permissions are 755 (all permissions for owner and read and execute permissions for all others). Also, the directory itself should be write-protected from others.

   For example:

   a. Move to the `/var/adm/acct` directory by typing:

```
cd /var/adm/acct
```

   b. Change the ownership for the `sum`, `nite`, and `fiscal` directories to **adm** group authority by typing:

```
chown adm sum/* nite/* fiscal/*
```

   To prevent tampering by users trying to avoid charges, deny write permission for others on these files. Change the **accton** command group owner to **adm**, and permissions to 710, that is, no permissions for others. Processes owned by **adm** can execute the **accton** command, but ordinary users cannot.

3. The `/var/adm/wtmp` file must also be owned by **adm**. If `/var/adm/wtmp` is owned by root, you will see the following message during startup:

```
/var/adm/acct/startup: /var/adm/wtmp: Permission denied
```

   To correct the ownership of `/var/adm/wtmp`, change ownership to the **adm** group by typing the following command:

```
chown adm /var/adm/wtmp
```

Fixing Accounting errors

You can correct date and time-stamp inconsistencies.

You must have root user or adm group authority to perform this procedure.

Processing the `/var/adm/wtmp` file might produce some warnings mailed to root. The `wtmp` file contains information collected by `/etc/init` and `/bin/login` and is used by accounting scripts primarily for calculating connect time (the length of time a user is logged in). Unfortunately, date changes confuse the program that processes the `wtmp` file. As a result, the **runacct** command sends mail to root and adm complaining of any errors after a date change since the last time accounting was run.

1. Determine if you received any errors.

The **acctcon1** command outputs error messages that are mailed to adm and root by the **runacct** command.

For example, if the **acctcon1** command stumbles after a date change and fails to collect connect times, adm might get mail like the following mail message:

```
Mon Jan 6 11:58:40 CST 1992
acctcon1: bad times: old: Tue Jan 7 00:57:14 1992
new: Mon Jan 6 11:57:59 1992
acctcon1: bad times: old: Tue Jan 7 00:57:14 1992
new: Mon Jan 6 11:57:59 1992
acctcon1: bad times: old: Tue Jan 7 00:57:14 1992
new: Mon Jan 6 11:57:59 1992
```

2. Adjust the wtmp file by typing:

```
/usr/sbin/acct/wtmpfix wtmp
```

The **wtmpfix** command examines the wtmp file for date and time-stamp inconsistencies and corrects problems that could make **acctcon1** fail. However, some date changes slip by **wtmpfix**.

3. Run accounting right before shutdown or immediately after startup.

Using the **runacct** command at these times minimizes the number of entries with bad times. The **runacct** command continues to send mail to the root and adm accounts, until you edit the **runacct** script, find the WTMPFIX section, and comment out the line where the file log gets mailed to the root and adm accounts.

Related tasks

Fixing wtmp errors

The /var/adm/wtmp, or "who temp" file, might cause problems in the day-to-day operation of the accounting system. You can fix wtmp errors.

Accounting errors encountered when running the runacct command

You might encounter errors when running the **runacct** command.

Note: You must have root user or adm group authority to run the **runacct** command.

The **runacct** command processes files that are often very large. The procedure involves several passes through certain files and consumes considerable system resources while it is taking place. Since the **runacct** command consumes considerable resources, it is normally run early in the morning when it can take over the machine and not disturb anyone.

The **runacct** command is a script divided into different stages. The stages allow you to restart the command where it stopped, without having to rerun the entire script.

When the **runacct** encounters problems, it sends error messages to different destinations depending on where the error occurred. Usually it sends a date and a message to the console directing you to look in the active*MMDD* file (such as active0621 for June 21st) which is in the /usr/adm/acct/nite directory. When the **runacct** command aborts, it moves the entire active file to active*MMDD* and appends a message describing the problem.

Review the following error message tables for errors that you might encounter when running the **runacct** command.

Note:

- The abbreviation *MMDD* stands for the month and day, such as 0102 for January 2. For example, an unrecoverable error during the CONNECT1 process on January 2 creates the file active0102 containing the error message.
- The abbreviation "SE message" stands for the standard error message such as:

```
********* ACCT ERRORS : see active0102 *********
```

Preliminary State and Error Messages from the **runnacct** Command				
State	**Command**	**Unrecoverable?**	**Error Message**	**Destinations**
pre	**runacct**	yes	* 2 CRONS or ACCT PROBLEMS * ERROR: locks found, run aborted	console, mail, active
pre	**runacct**	yes	runacct: Insufficient space in /usr (*nnn* blks); Terminating procedure	console, mail, active
pre	**runacct**	yes	SE message; ERROR: acctg already run for 'date': check lastdate	console, mail, active*MMDD*
pre	**runacct**	no	* SYSTEM ACCOUNTING STARTED *	console
pre	**runacct**	no	restarting acctg for 'date' at STATE	console active, console
pre	**runacct**	no	restarting acctg for 'date' at state (argument $2) previous state was STATE	active
pre	**runacct**	yes	SE message; Error: runacct called with invalid arguments	console, mail, active*MMDD*

States and Error Messages from the **runacct** Command				
State	**Command**	**Unrecoverable?**	**Error Message**	**Destinations**
SETUP	**runacct**	no	ls -l fee pacct* /var/adm/wtmp	active
SETUP	**runacct**	yes	SE message; ERROR: turnacct switch returned rc=error	console, mail, active*MMDD*

States and Error Messages from the **runacct** Command *(continued)*				
State	**Command**	**Unrecoverable ?**	**Error Message**	**Destinations**
SETUP	**runacct**	yes	SE message; ERROR: SpacctMMDD already exists file setups probably already run	active*MMDD*
SETUP	**runacct**	yes	SE message; ERROR: wtmpMMDD already exists: run setup manually	console, mail, active*MMDD*
WTMPFIX	**wtmpfix**	no	SE message; ERROR: wtmpfix errors see xtmperrorMMDD	active*MMDD*, wtmperror*MMDD*
WTMPFIX	**wtmpfix**	no	wtmp processing complete	active
CONNECT1	**acctcon1**	no	SE message; (errors from acctcon1 log)	console, mail, active*MMDD*
CONNECT2	**acctcon2**	no	connect acctg complete	active
PROCESS	**runacct**	no	WARNING: accounting already run for pacct*N*	active
PROCESS	**acctprc1 acctprc2**	no	process acctg complete for SpacctNMMDD	active
PROCESS	**runacct**	no	all process actg complete for date	active
MERGE	**acctmerg**	no	tacct merge to create dayacct complete	active
FEES	**acctmerg**	no	merged fees OR no fees	active
DISK	**acctmerg**	no	merged disk records OR no disk records	active
MERGEACCT	**acctmerg**	no	WARNING: recreating sum/tacct	active

States and Error Messages from the **runacct** Command *(continued)*				
State	**Command**	**Unrecoverable ?**	**Error Message**	**Destinations**
MERGEACCT	**acctmerg**	no	`updated sum/ tacct`	active
CMS	**runacct**	no	`WARNING: recreating sum/cms`	active
CMS	**acctcms**	no	`command summaries complete`	active
CLEANUP	**runacct**	no	`system accounting completed at 'date'`	active
CLEANUP	**runacct**	no	`*SYSTEM ACCOUNTING COMPLETED*`	console
<wrong>	**runacct**	yes	`SE message; ERROR: invalid state, check STATE`	console, mail, active*MMDD*

Note: The label <wrong> in the previous table does not represent a state, but rather a state other than the correct state that was written in the state file `/usr/adm/acct/nite/statefile`.

Summary of Message Destinations	
Destination	**Description**
console	The `/dev/console` device
mail	Message mailed to **root** and **adm** accounts
active	The `/usr/adm/acct/nite/active` file
activeMMDD	The `/usr/adm/acct/nite/active`*MMDD* file
wtmperrMMDD	The `/usr/adm/acct/nite/wtmperror`*MMDD* file
STATE	Current state in `/usr/adm/acct/nite/statefile` file
fd2log	Any other error messages

System Resource Controller

The System Resource Controller (SRC) provides a set of commands and subroutines to make it easier for the system manager and programmer to create and control subsystems.

A *subsystem* is any program or process or set of programs or processes that is usually capable of operating independently or with a controlling system. A subsystem is designed as a unit to provide a designated function.

The SRC was designed to minimize the need for operator intervention. It provides a mechanism to control subsystem processes using a common command line and the C interface. This mechanism includes the following:

- Consistent user interface for start, stop, and status inquiries

- Logging of the abnormal termination of subsystems
- Notification program called at the abnormal system termination of related processes
- Tracing of a subsystem, a group of subsystems, or a subserver
- Support for control of operations on a remote system
- Refreshing of a subsystem (such as after a configuration data change).

The SRC is useful if you want a common way to start, stop, and collect status information on processes.

Related concepts

Introduction to AIX for BSD system managers

The following are tips to help Berkeley Software Distribution (BSD) system managers get started managing AIX.

Subsystem components

The following are the properties and components of a subsystem.

A subsystem can have one or more of the following properties:

- Is known to the system by name
- Requires a more complex execution environment than a subroutine or nonprivileged program
- Includes application programs and libraries as well as subsystem code
- Controls resources that can be started and stopped by name
- Requires notification if a related process is unsuccessful to perform cleanup or to recover resources
- Requires more operational control than a simple daemon process
- Needs to be controlled by a remote operator
- Implements subservers to manage specific resources
- Does not put itself in the background.

A few subsystem examples are ypserv, ntsd, qdaemon, inetd, syslogd, and sendmail.

Note: See each specific subsystem for details of its SRC capabilities.

Use the **lssrc -a** command to list active and inactive subsystems on your system.

The following defines subsystem groups and subservers:

Subsystem Group

A *subsystem group* is a group of any specified subsystems. Grouping subsystems together allows the control of several subsystems at one time. A few subsystem group examples are TCP/IP, SNA Services, Network Information System (NIS), and Network File Systems (NFS).

Subserver

A *subserver* is a program or process that belongs to a subsystem. A subsystem can have multiple subservers and is responsible for starting, stopping, and providing status of subservers. Subservers can be defined only for a subsystem with a communication type of IPC message queues and sockets. Subsystems using signal communications do not support subservers.

Subservers are started when their parent subsystems are started. If you try to start a subserver and its parent subsystem is not active, the **startsrc** command starts the subsystem as well.

SRC hierarchy

The System Resource Controller hierarchy begins with the operating system followed by a subsystem group (such as **tcpip**), which contains a subsystem (such as the **inetd** daemon), which in turn can own several subservers (such as the **ftp** daemon and the **finger** command).

SRC administration commands

You can administer SRC from the command line.

The SRC administration commands are:

Item	Description
srcmstr daemon	Starts the System Resource Controller
startsrc command	Starts a subsystem, subsystem group, or subserver
stopsrc command	Stops a subsystem, subsystem group, or subserver
refresh command	Refreshes a subsystem
traceson command	Turns on tracing of a subsystem, a group of subsystems, or a subserver
tracesoff command	Turns off tracing of a subsystem, a group of subsystems, or a subserver
lssrc command	Gets status on a subsystem.

Starting the System Resource Controller

The System Resource Controller (SRC) is started during system initialization with a record for the `/usr/sbin/srcmstr` daemon in the `/etc/inittab` file.

The following are the prerequisites for starting the SRC:

- Reading and writing the `/etc/inittab` file requires root user authority.
- The **mkitab** command requires root user authority.
- The **srcmstr** daemon record must exist in the `/etc/inittab` file.

The default `/etc/inittab` file already contains such a record, so this procedure might be unnecessary. You can also start the SRC from the command line, a profile, or a shell script, but there are several reasons for starting it during initialization:

- Starting the SRC from the `/etc/inittab` file allows the **init** command to restart the SRC if it stops for any reason.
- The SRC is designed to simplify and reduce the amount of operator intervention required to control subsystems. Starting the SRC from any source other than the `/etc/inittab` file is counterproductive to that goal.
- The default `/etc/inittab` file contains a record for starting the print scheduling subsystem (**qdaemon**) with the **startsrc** command. Typical installations have other subsystems started with **startsrc** commands in the `/etc/inittab` file as well. Because the **srcmstr** command requires the SRC be running, removing the **srcmstr** daemon from the `/etc/inittab` file causes these **startsrc** commands to fail.

Note: This procedure is necessary only if the `/etc/inittab` file does not already contain a record for the **srcmstr** daemon.

1. Make a record for the **srcmstr** daemon in the `/etc/inittab` file using the **mkitab** command. For example, to make a record identical to the one that appears in the default `/etc/inittab` file, type:

   ```
   mkitab -i fbcheck srcmstr:2:respawn:/usr/sbin/srcmstr
   ```

   The **-i fbcheck** flag ensures that the record is inserted before all subsystems records.

2. Tell the **init** command to reprocess the /etc/inittab file by typing:

```
telinit q
```

   When **init** revisits the /etc/inittab file, it processes the newly entered record for the **srcmstr** daemon and starts the SRC.

Related concepts

Subsystem control

The **traceson** command can be used to turn on, and the **traceoff** command can be used to turn off, tracing of a System Resource Controller (SRC) resource such as a subsystem, a group of subsystems, or a subserver.

Related tasks

Refreshing a subsystem or subsystem group

Use the **refresh** command to tell a System Resource Controller (SRC) resource such as a subsystem or a group of subsystems to refresh itself.

Starting or stopping a subsystem, subsystem group, or subserver

Use the **startsrc** command to start a System Resource Controller (SRC) resource such as a subsystem, a group of subsystems, or a subserver. Use the **stopsrc** command to stop an SRC resource such as a subsystem, a group of subsystems, or a subserver.

The following are the prerequisites for starting or stopping a subsystem, subsystem group, or subserver:

- To start or stop an SRC resource, the SRC must be running. The SRC is normally started during system initialization. The default /etc/inittab file, which determines what processes are started during initialization, contains a record for the **srcmstr** daemon (the SRC). To see if the SRC is running, type ps -A and look for a process named srcmstr.
- The user or process starting an SRC resource must have root user authority. The process that initializes the system (**init** command) has root user authority.
- The user or process stopping an SRC resource must have root user authority.

The **startsrc** command can be used:

- From the /etc/inittab file so the resource is started during system initialization
- From the command line
- With SMIT

When you start a subsystem group, all of its subsystems are also started. When you start a subsystem, all of its subservers are also started. When you start a subserver, its parent subsystem is also started if it is not already running.

When you stop a subsystem, all its subservers are also stopped. However, when you stop a subserver, the state of its parent subsystem is not changed.

Both the **startsrc** and **stopsrc** commands contain flags that allow requests to be made on local or remote hosts. See the **srcmstr** command for the configuration requirements to support remote SRC requests.

Starting or stopping a subsystem tasks		
Task	***SMIT fast path***	***Command or file***
Start a subsystem	**smit startssys**	/bin/startsrc -s *SubsystemName* , or edit /etc/inittab
Stop a subsystem	**smit stopssys**	/bin/stopsrc -s *SubsystemName*

Related information
stopsrc command
startsrc command
srcmstr command

Displaying status of a subsystem or subsystems

Use the **lssrc** command to display the status of a System Resource Controller (SRC) resource such as a subsystem, a group of subsystems, or a subserver.

All subsystems can return a short status report that includes which group the subsystem belongs to, whether the subsystem is active, and what its process ID (PID) is. If a subsystem does not use the signals communication method, it can be programmed to return a long status report containing additional status information.

The **lssrc** command provides flags and parameters for specifying the subsystem by name or PID, for listing all subsystems, for requesting a short or long status report, and for requesting the status of SRC resources either locally or on remote hosts.

See the **srcmstr** command for the configuration requirements to support remote SRC requests.

Displaying the Status of Subsystems Tasks		
Task	***SMIT Fast Path***	***Command or File***
Display the status of a subsystem (long format)	**smit qssys**	**lssrc -l -s** *SubsystemName*
Display the status of all subsystems	**smit lsssys**	**lssrc -a**
Display the status of all subsystems on a particular host		**lssrc -h***HostName* **-a**

Refreshing a subsystem or subsystem group

Use the **refresh** command to tell a System Resource Controller (SRC) resource such as a subsystem or a group of subsystems to refresh itself.

The following are the prerequisites for refreshing a subsystem or subsystem group:

- The SRC must be running.
- The resource you want to refresh must not use the signals communications method.
- The resource you want to refresh must be programmed to respond to the refresh request.

The **refresh** command provides flags and parameters for specifying the subsystem by name or PID. You can also use it to request a subsystem or group of subsystems be refreshed, either locally or on remote hosts. See the **srcmstr** command for the configuration requirements to support remote SRC requests.

Refreshing a Subsystem or Subsystem Group		
Task	**SMIT Fast Path**	**Command or File**
Refresh a Subsystem	**smit refresh**	**refresh -s Subsystem**

Related tasks
Starting the System Resource Controller

The System Resource Controller (SRC) is started during system initialization with a record for the /usr/sbin/srcmstr daemon in the /etc/inittab file.

Subsystem control

The **traceson** command can be used to turn on, and the **traceoff** command can be used to turn off, tracing of a System Resource Controller (SRC) resource such as a subsystem, a group of subsystems, or a subserver.

Use the **traceson** command to turn on tracing of a System Resource Controller (SRC) resource such as a subsystem, a group of subsystems, or a subserver.

Use the **tracesoff** command to turn off tracing of a System Resource Controller (SRC) resource such as a subsystem, a group of subsystems, or a subserver.

The **traceson** and **traceoff** commands can be used to remotely turn on or turn off tracing on a specific host. See the **srcmstr** command for the configuration requirements for supporting remote SRC requests.

Prerequisites

- To turn the tracing of an SRC resource either on or off , the SRC must be running.
- The resource you want to trace must not use the signals communications method.
- The resource you want to trace must be programmed to respond to the trace request.

Turning On/Off Subsystem, Subsystem Group, or Subserver Tasks		
Task	***SMIT Fast Path***	***Command or File***
Turn on Subsystem Tracing (short format)	**smit tracessyson**	**traceson -s Subsystem**
Turn on Subsystem Tracing (long format)	**smit tracessyson**	**traceson -l -s Subsystem**
Turn off Subsystem Tracing	**smit tracessysoff**	**tracesoff -s Subsystem**

Related tasks

Starting the System Resource Controller

The System Resource Controller (SRC) is started during system initialization with a record for the /usr/sbin/srcmstr daemon in the /etc/inittab file.

Operating system files

Files are used for all input and output (I/O) of information in the operating system, to standardize access to both software and hardware.

Input occurs when the contents of a file is modified or written to. *Output* occurs when the contents of one file is read or transferred to another file. For example, to create a printed copy of a file, the system reads the information from the text file and writes that information to the file representing the printer.

Types of files

The types of files recognized by the system are either **regular**, **directory**, or **special**. However, the operating system uses many variations of these basic types.

The following basic types of files exist:

Item	Description
regular	Stores data (text, binary, and executable)
directory	Contains information used to access other files
special	Defines a FIFO (first-in, first-out) pipe file or a physical device

All file types recognized by the system fall into one of these categories. However, the operating system uses many variations of these basic types.

Regular files

Regular files are the most common files and are used to contain data. Regular files are in the form of text files or binary files:

Text files

Text files are regular files that contain information stored in ASCII format text and are readable by the user. You can display and print these files. The lines of a text file must not contain NUL characters, and none can exceed {LINE_MAX} bytes in length, including the newline character.

The term *text file* does not prevent the inclusion of control or other nonprintable characters (other than NUL). Therefore, standard utilities that list text files as inputs or outputs are either able to process the special characters or they explicitly describe their limitations within their individual sections.

Binary files

Binary files are regular files that contain information readable by the computer. Binary files might be executable files that instruct the system to accomplish a job. Commands and programs are stored in executable, binary files. Special compiling programs translate ASCII text into binary code.

Text and binary files differ only in that text files have lines of less than {LINE_MAX} bytes, with no NUL characters, each terminated by a newline character.

Directory files

Directory files contain information that the system needs to access all types of files, but directory files do not contain the actual file data. As a result, directories occupy less space than a regular file and give the file system structure flexibility and depth. Each directory entry represents either a file or a subdirectory. Each entry contains the name of the file and the file's index node reference number (*i-node number*). The i-node number points to the unique index node assigned to the file. The i-node number describes the location of the data associated with the file. Directories are created and controlled by a separate set of commands.

Special files

Special files define devices for the system or are temporary files created by processes. The basic types of special files are FIFO (first-in, first-out), block, and character. FIFO files are also called *pipes*. Pipes are created by one process to temporarily allow communication with another process. These files cease to exist when the first process finishes. Block and character files define devices.

Every file has a set of permissions (called *access modes*) that determine who can read, modify, or execute the file.

Related concepts

File and directory access modes

Every file has an owner. For new files, the user who creates the file is the owner of that file. The owner assigns an *access mode* to the file. Access modes grant other system users permission to read, modify, or execute the file. Only the file's owner or users with root authority can change the access mode of a file.

File naming conventions

The name of each file must be unique within the directory where it is stored. This ensures that the file also has a unique path name in the file system.

File naming guidelines are:

- A file name can be up to 255 characters long and can contain letters, numbers, and underscores.
- The operating system is case-sensitive, which means it distinguishes between uppercase and lowercase letters in file names. Therefore, FILEA, FiLea, and filea are three distinct file names, even if they reside in the same directory.

- File names should be as descriptive and meaningful as possible.
- Directories follow the same naming conventions as files.
- Certain characters have special meaning to the operating system. Avoid using these characters when you are naming files. These characters include the following:

```
/ \ " ' * ; - ? [ ] ( ) ~ ! $ { } &lt > # @ & | space tab newline
```

- A file name is hidden from a normal directory listing if it begins with a dot (`.`). When the **`ls`** command is entered with the **-a** flag, the hidden files are listed along with regular files and directories.

File path names

The path name for each file and directory in the file system consists of the names of every directory that precedes it in the tree structure.

Because all paths in a file system originate from the /(`root`) directory, each file in the file system has a unique relationship to the root directory, known as the *absolute path name*. Absolute path names begin with the slash (/) symbol. For example, the absolute path name of file h could be `/B/C/h`. Notice that two files named h can exist in the system. Because the absolute paths to the two files are different, `/B/h` and `/B/C/h`, each file named h has a unique name within the system. Every component of a path name is a directory except the final component. The final component of a path name can be a file name.

Note: Path names cannot exceed 1023 characters in length.

Pattern matching with wildcards and metacharacters

Wildcard characters provide a convenient way to specify multiple file names or directory names.

The wildcard characters are asterisk (*) and question mark (?). The metacharacters are open and close square brackets (`[ ]`), hyphen (-), and exclamation mark (`!`).

*Pattern matching using the * wildcard character*
Use the asterisk (*) to match any sequence or string of characters.

The (*) indicates any characters, including no characters.

See the following examples:

- If you have the following files in your directory:

```
1test 2test afile1 afile2 bfile1 file file1 file10 file2 file3
```

  and you want to refer to only the files that begin with `file`, use:

```
file*
```

  The files selected would be: `file`, `file1`, `file10`, `file2`, and `file3`.
- To refer to only the files that contain the word `file`, use:

```
*file*
```

  The files selected would be: `afile1`, `afile2`, `bfile1`, `file`, `file1`, `file10`, `file2`, and `file3`.

Pattern matching using the ? wildcard character
Use the ? to match any one character.

The ? indicates any single character. See the following examples:

- To refer to only the files that start with **file** and end with a single character, use:

```
file?
```

  The files selected would be: `file1`, `file2`, `file3`.

- To refer to only the files that start with **file** and end with any two characters, use:

```
file??
```

  The file selected would be: `file10`.

Pattern matching using [ ] shell metacharacters
Metacharacters offer another type of wildcard notation by enclosing the desired characters within `[ ]`. It is like using the ?, but it allows you to choose specific characters to be matched.

The `[ ]` also allow you to specify a range of values using the hyphen (-). To specify all the letters in the alphabet, use `[[:alpha:]]`. To specify all the lowercase letters in the alphabet, use `[[:lower:]]`.

See the following examples:

- To refer to only the files that end in 1 or 2, use:

```
*file[12]
```

  The files selected would be: `afile1`, `afile2`, `file1`, and `file2`.
- To refer to only the files that start with any number, use:

```
[0123456789]* or [0-9]*
```

  The files selected would be: `1test` and `2test`.
- To refer to only the files that do not begin with an a, use:

```
[!a]*
```

  The files selected would be: `1test`, `2test`, `bfile1`, `file`, `file1`, `file10`, `file2`, and `file3`.

Pattern matching versus regular expressions

Regular expressions allow you to select specific strings from a set of character strings. The use of regular expressions is generally associated with text processing.

Regular expressions can represent a wide variety of possible strings. While many regular expressions can be interpreted differently depending on the current locale, globalization features provide for contextual invariance across locales.

Pattern matching is used by the shell commands such as the **ls** command, whereas regular expressions are used to search for strings of text in a file by using commands, such as the **grep** command.

The following table lists some of the equivalent regular expressions for corresponding pattern matching characters:

Table 57. Difference between Pattern matching and Regular Expression

Pattern Matching	Regular Expression
*	.*
?	.
[!a]	[^a]
[abc]	[abc]
[[:alpha:]]	[[:alpha:]]

The following table lists some of the examples for command, output, and descriptions:

Table 58. Examples of commands with pattern matching and regular expressions		
Command	**Command Output**	**Explanation**
# ls	`1test 2test afile1 afile2 bfile1 file file1 file2 file3 file10`	Lists all the files in the directory.
# ls file?	`file file1 file2 file3 file10`	Uses pattern matching to list all files that start with 'file'.
# ls | grep 'file?'	`No output`	Uses regular expression to find the specific files. However, no output is displayed because the question mark symbol (?) is not a valid regular expression character.
# ls | grep 'file.'	`file file1 file2 file3 file10`	Uses a valid regular expression to search the files that start with 'file'.
# ls *af*	`afile1 afile2`	Uses pattern matching to list all files that contain 'af'.
# ls | grep '*af*'	`No output`	Uses regular expression to find the specific files. However, no output is displayed because of an invalid regular expression.
# ls | grep '.*af.*'	`afile1 afile2`	Uses a valid regular expression to search the files that contain 'af'.

Related information
awk command

Administering files

There are many ways to work with the files on your system. Usually you create a text file with a text editor.

The common editors in the UNIX environment are `vi` and ed. Because several text editors are available, you can choose an editor you feel comfortable with.

You can also create files by using input and output redirection. You can send the output of a command to a new file or append it to an existing file.

After creating and modifying files, you might have to copy or move files from one directory to another, rename files to distinguish different versions of a file, or give different names to the same file. You might also need to create directories when working on different projects.

Also, you might need to delete certain files. Your directory can quickly get cluttered with files that contain old or useless information. To release storage space on your system, ensure that you delete files that are no longer needed.

Deleting files (rm command)

Use the **rm** command to remove files you no longer need.

The **rm** command removes the entries for a specified file, group of files, or certain select files from a list within a directory. User confirmation, read permission, and write permission are not required before a file is removed when you use the **rm** command. However, you must have write permission for the directory containing the file.

The following are examples of how to use the **rm** command:

- To delete the file named `myfile`, type the following:

```
rm myfile
```

- To delete all the files in the `mydir` directory, one by one, type the following:

```
rm -i mydir/*
```

  After each file name displays, type y and press Enter to delete the file. Or to keep the file, just press Enter.

Moving and renaming files (mv command)

Use the **mv** command to move files and directories from one directory to another or to rename a file or directory. If you move a file or directory to a new directory without specifying a new name, it retains its original name.

Attention: The **mv** command can overwrite many existing files unless you specify the **-i** flag. The **-i** flag prompts you to confirm before it overwrites a file. The **-f** flag does not prompt you. If both the **-f** and **-i** flags are specified in combination, the last flag specified takes precedence.

Moving files with mv command

The following are examples of how to use the **mv** command:

- To move a file to another directory and give it a new name, type the following:

```
mv intro manual/chap1
```

  This moves the `intro` file to the `manual/chap1` directory. The name **intro** is removed from the current directory, and the same file appears as `chap1` in the `manual` directory.

- To move a file to another directory, keeping the same name, type the following:

```
mv chap3 manual
```

  This moves `chap3` to `manual/chap3`.

Renaming files with mv command

Use the **mv** command to change the name of a file without moving it to another directory.

To rename a file, type the following:

```
mv appendix apndx.a
```

This renames the `appendix` file to `apndx.a`. If a file named `apndx.a` already exists, its old contents are replaced with those of the `appendix` file.

Copying files (cp command)

Use the **cp** command to create a copy of the contents of the file or directory specified by the *SourceFile* or *SourceDirectory* parameters into the file or directory specified by the *TargetFile* or *TargetDirectory* parameters.

If the file specified as the *TargetFile* exists, the copy writes over the original contents of the file without warning. If you are copying more than one *SourceFile*, the target must be a directory.

If a file with the same name exists at the new destination, the copied file overwrites the file at the new destination. Therefore, it is a good practice to assign a *new* name for the copy of the file to ensure that a file of the same name does not exist in the destination directory.

To place a copy of the *SourceFile* into a directory, specify a path to an existing directory for the *TargetDirectory* parameter. Files maintain their respective names when copied to a directory unless you specify a new file name at the end of the path. The **cp** command also copies entire directories into other directories if you specify the **-r** or **-R** flags.

You can also copy special-device files using the **-R** flag. Specifying **-R** causes the special files to be re-created under the new path name. Specifying the **-r** flag causes the **cp** command to attempt to copy the special files to regular files.

The following are examples of how to use the **cp** command:

- To make a copy of a file in the current directory, type the following:

```
cp prog.c prog.bak
```

  This copies `prog.c` to `prog.bak`. If the `prog.bak` file does not already exist, then the **cp** command creates it. If it does exist, then the **cp** command replaces it with a copy of the `prog.c` file.
- To copy a file in your current directory into another directory, type the following:

```
cp jones /home/nick/clients
```

  This copies the `jones` file to `/home/nick/clients/jones`.
- To copy all the files in a directory to a new directory, type the following:

```
cp /home/janet/clients/* /home/nick/customers
```

  This copies only the files in the `clients` directory to the `customers` directory.
- To copy a specific set of files to another directory, type the following:

```
cp jones lewis smith /home/nick/clients
```

  This copies the `jones`, `lewis`, and `smith` files in your current working directory to the `/home/nick/clients` directory.
- To use pattern-matching characters to copy files, type the following:

```
cp programs/*.c .
```

  This copies the files in the `programs` directory that end with `.c` to the current directory, indicated by the single dot (`.`). You must type a space between the c and the final dot.

Finding files (find command)

Use the **find** command to recursively search the directory tree for each specified *Path*, seeking files that match a Boolean expression written using the terms given in the following text.

The output from the **find** command depends on the terms specified by the *Expression* parameter.

The following are examples of how to use the `find` command:

- To list all files in the file system with the name `.profile`, type the following:

```
find / -name .profile
```

  This searches the entire file system and writes the complete path names of all files named `.profile`. The slash (/) tells the **find** command to search the /(`root`) directory and all of its subdirectories.

  To save time, limit the search by specifying the directories where you think the files might be.
- To perform a case-insensitive search operation for all the files with the `test` filename in the current directory and all of its subdirectories, enter the following command:

```
find . -iname test
```

- To list files having a specific permission code of `0600` in the current directory tree, type the following:

```
find . -perm 0600
```

This lists the names of the files that have *only* owner-read and owner-write permission. The dot (`.`) tells the **find** command to search the current directory and its subdirectories. For an explanation of permission codes, see the **chmod** command.

- To search several directories for files with certain permission codes, type the following:

```
find manual clients proposals -perm -0600
```

This lists the names of the files that have owner-read and owner-write permission and possibly other permissions. The `manual`, `clients`, and `proposals` directories and their subdirectories are searched. In the previous example, `-perm 0600` selects only files with permission codes that match `0600` exactly. In this example, `-perm -0600` selects files with permission codes that allow the accesses indicated by `0600` and other accesses above the `0600` level. This also matches the permission codes 0622 and 2744.

- To list all files in the current directory that have been changed during the current 24-hour period, type the following:

```
find . -ctime 1
```

- To search for regular files with multiple links, type the following:

```
find . -type f -links +1
```

This lists the names of the ordinary files (`-type f`) that have more than one link (`-links +1`).

Note: Every directory has at least two links: the entry in its parent directory and its own `.`(dot) entry. For more information on multiple file links, see the **ln** command.

- To search for all files that are exactly 414 bytes in length, type the following:

```
find . -size 414c
```

- >| To perform a case-insensitive search operation for all the files in the root directory and its subdirectories whose filename contains the string `main` and the extension is of any length that ends with the alphabet o, enter the following command:

```
find / -iname "*main*.*o"
```

The search result of this command returns files like `main.o`, `app_main.c.o`, `mAin.O`, or `APP_MAIN.o`. |<

- >| To perform a case-insensitive search operation for all the files in the root directory and its subdirectories whose filename contains the string `main` and have a single-character extension, enter the following command:

```
find / -iname "*main*.?"
```

The search result of this command returns files like `main.c`, `app_main.o`, `MAIN.c`, or `App_main.o`. |<

- >| To perform a case-insensitive search operation for all the files in the root directory and its subdirectories whose extension is `.T`, followed by any one character, and an alphabet `t`, enter the following command:

```
find / -iname "*.T?t"
```

The search result of this command returns files like `file.T1t`, `log.Tot`, `file.txt`, `log.tot`, or `log.toT`. |<

- >| To perform a case-insensitive search operation for all the files in the root directory and its subdirectories whose extension is either `.Sh`, `.md`, `.Sd`, or `.mh`, enter the following command:

```
find / -iname "*.[Sm][hd]"
```

The search result of this command returns files like `readme.sh`, `logo.md`, `logo.sd`, `readme.mh`, `readme.SH`, or `logo.mD`.|<

Displaying the file type (file command)

Use the **file** command to read the files specified by the *File* or **-f***FileList* parameter, perform a series of tests on each one, and attempt to classify the files by type. The command then writes the file types to standard output.

If a file appears to be ASCII, the **file** command examines the first 512 bytes and determines its language. If a file does not appear to be ASCII, the **file** command further attempts to determine whether it is a binary data file or a text file that contains extended characters.

If the *File* parameter specifies an executable or object module file and the version number is greater than 0, the **file** command displays the version stamp.

The **file** command uses the `/etc/magic` file to identify files that have a magic number; that is, any file containing a numeric or string constant that indicates the type.

The following are examples of how to use the **file** command:

- To display the type of information the file named `myfile` contains, type the following:

  ```
  file myfile
  ```

  This displays the file type of `myfile` (such as directory, data, ASCII text, C program source, or archive).
- To display the type of each file named in the `filenames.lst` file, which contains a list of file names, type the following:

  ```
  file -f filenames.lst
  ```

  This displays the type of each file named in the `filenames.lst` file. Each file name must display on a separate line.
- To create the `filenames.lst` file that contains all the file names in the current directory, type the following:

  ```
  ls > filenames.lst
  ```

  Edit the `filenames.lst` file as desired.

Commands for displaying file contents (pg, more, page, and cat commands)

The **pg**, **more**, and **page** commands allow you to view the contents of a file and control the speed at which your files are displayed.

You can also use the **cat** command to display the contents of one or more files on your screen. Combining the **cat** command with the **pg** command allows you to read the contents of a file one full screen at a time.

You can also display the contents of files by using input and output redirection.

Related concepts

Input and output redirection

The AIX operating system allows you to manipulate the input and output (I/O) of data to and from your system by using specific I/O commands and symbols.

Using the pg command

Use the **pg** command to read the files named in the **File** parameter and writes them to standard output one screen at a time.

If you specify hyphen (-) as the **File** parameter or run the **pg** command without options, the **pg** command reads standard input. Each screen is followed by a prompt. If you press the Enter key, another screen displays. Subcommands used with the **pg** command let you review content that has already passed.

For example, to look at the contents of the file `myfile` one page at a time, type the following:

```
pg myfile
```

Using the more or page commands
Use the **more** or **page** command to display continuous text one screen at a time.

It pauses after each screen and prints the *filename* and percent completed (for example, `myfile (7%)`) at the bottom of the screen. If you then press the Enter key, the **more** command displays an additional line. If you press the Spacebar, the **more** command displays another screen of text.

Note: On some terminal models, the **more** command clears the screen, instead of scrolling, before displaying the next screen of text.

For example, to view a file named `myfile`, type the following:

```
more myfile
```

Press the Spacebar to view the next screen.

cat command
Use the **cat** command to read each *File* parameter in sequence and writes it to standard output.

See the following examples:

- To display the contents of the file `notes`, type the following:

  ```
  cat notes
  ```

  If the file is more than 24 lines long, some of it scrolls off the screen. To list a file one page at a time, use the **pg** command.
- To display the contents of the files `notes`, `notes2`, and `notes3`, type the following:

  ```
  cat notes notes2 notes3
  ```

Finding text strings within files (grep command)
Use the **grep** command to search the specified file for the pattern specified by the *Pattern* parameter and writes each matching line to standard output.

The following are examples of how to use the **grep** command:

- To search in a file named `pgm.s` for a pattern that contains some of the pattern-matching characters *, ^, ?, [,], \(, \), \{, and \}, in this case, lines starting with any lowercase or uppercase letter, type the following:

  ```
  grep "^[a-zA-Z]" pgm.s
  ```

  This displays all lines in the `pgm.s` file that begin with a letter.
- To display all lines in a file named `sort.c` that do not match a particular pattern, type the following:

  ```
  grep -v bubble sort.c
  ```

  This displays all lines that do not contain the word `bubble` in the `sort.c` file.
- To display lines in the output of the **ls** command that match the string `staff`, type the following:

  ```
  ls -l | grep staff
  ```

Sorting text files (sort command)

Use the **sort** command to alphabetize lines in the files specified by the **File** parameters and write the result to standard output.

If the **File** parameter specifies more than one file, the **sort** command concatenates the files and alphabetizes them as one file.

Note: The **sort** command is case-sensitive and orders uppercase letters before lowercase (this behavior is dependent on the locale).

In the following examples, the contents of the file named `names` are:

```
marta
denise
joyce
endrica
melanie
```

and the contents of the file named `states` are:

```
texas
colorado
ohio
```

- To display the sorted contents of the file named `names`, type the following:

  ```
  sort names
  ```

  The system displays information similar to the following:

  ```
  denise
  endrica
  joyce
  marta
  melanie
  ```

- To display the sorted contents of the `names` and `states` files, type the following:

  ```
  sort names states
  ```

  The system displays information similar to the following:

  ```
  colorado
  denise
  endrica
  joyce
  marta
  melanie
  ohio
  texas
  ```

- To replace the original contents of the file named `names` with its sorted contents, type the following:

  ```
  sort -o names names
  ```

  This replaces the contents of the `names` file with the same data but in sorted order.

Comparing files (diff command)

Use the **diff** command to compare text files. It can compare single files or the contents of directories.

When the **diff** command is run on regular files, and when it compares text files in different directories, the **diff** command tells which lines must be changed in the files so that they match.

The following are examples of how to use the **diff** command:

- To compare two files, type the following:

  ```
  diff chap1.bak chap1
  ```

This displays the differences between the `chap1.bak` and `chap1` files.

- To compare two files while ignoring differences in the amount of white space, type the following:

```
diff -w prog.c.bak prog.c
```

If the two files differ only in the number of spaces and tabs between words, the **diff -w** command considers the files to be the same.

Counting words, lines, and bytes in files (wc command)

Use the **wc** command to count the number of lines, words, and bytes in the files specified by the *File* parameter.

If a file is not specified for the *File* parameter, standard input is used. The command writes the results to standard output and keeps a total count for all named files. If flags are specified, the ordering of the flags determines the ordering of the output. A *word* is defined as a string of characters delimited by spaces, tabs, or newline characters.

When files are specified on the command line, their names are printed along with the counts.

See the following examples:

- To display the line, word, and byte counts of the file named `chap1`, type the following:

```
wc chap1
```

This displays the number of lines, words, and bytes in the `chap1` file.

- To display only byte and word counts, type the following:

```
wc -cw chap*
```

This displays the number of bytes and words in each file where the name starts with `chap`, and displays the totals.

Displaying the first lines of files (head command)

Use the **head** command to write to standard output the first few lines of each of the specified files or of the standard input.

If no flag is specified with the **head** command, the first 10 lines are displayed by default.

For example, to display the first five lines of the `Test` file, type the following:

```
head -5 Test
```

Displaying the last lines of files (tail command)

Use the **tail** command to write the file specified by the *File* parameter to standard output beginning at a specified point.

See the following examples:

- To display the last 10 lines of the `notes` file, type the following:

```
tail notes
```

- To specify the number of lines to start reading from the end of the `notes` file, type the following:

```
tail -20 notes
```

- To display the `notes` file one page at a time, beginning with the 200th byte, type the following:

```
tail -c +200 notes | pg
```

- To follow the growth of the file named `accounts`, type the following:

```
tail -f accounts
```

This displays 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 you press the (Ctrl-C) key sequence to stop the display.

Cutting sections of text files (cut command)

Use the **cut** command to write selected bytes, characters, or fields from each line of a file to standard output.

See the following examples:

- To display several fields of each line of a file, type the following:

```
cut -f1,5 -d: /etc/passwd
```

  This displays the login name and full user name fields of the system password file. These are the first and fifth fields (`-f1,5`) separated by colons (`-d:`).

- If the `/etc/passwd` file looks like this:

```
su:*:0:0:User with special privileges:/:/usr/bin/sh
daemon:*:1:1::/etc:
bin:*:2:2::/usr/bin:
sys:*:3:3::/usr/src:
adm:*:4:4:system administrator:/var/adm:/usr/bin/sh
pierre:*:200:200:Pierre Harper:/home/pierre:/usr/bin/sh
joan:*:202:200:Joan Brown:/home/joan:/usr/bin/sh
```

  the **cut** command produces:

```
su:User with special privileges
daemon:
bin:
sys:
adm:system administrator
pierre:Pierre Harper
joan:Joan Brown
```

Pasting sections of text files (paste command)

Use the **paste** command to merge the lines of up to 12 files into one file.

See the following examples:

- If you have a file named `names` that contains the following text:

```
rachel
jerry
mark
linda
scott
```

  and another file named `places` that contains the following text:

```
New York
Austin
Chicago
Boca Raton
Seattle
```

  and another file named `dates` that contains the following text:

```
February 5
March 13
June 21
July 16
November 4
```

  To paste the text of the files `names`, places, and dates together, type the following:

```
paste names places dates > npd
```

This creates a file named npd that contains the data from the `names` file in one column, the `places` file in another, and the `dates` file in a third. The npd file now contains the following:

```
rachel          New York        February 5
jerry           Austin          March 13
mark            Chicago         June 21
linda           Boca Raton      July 16
scott           Seattle         November 4
```

A tab character separates the name, place, and date on each line. These columns do not align, because the tab stops are set at every eighth column.

- To separate the columns with a character other than a tab, type the following:

```
paste -d"!@" names places dates > npd
```

This alternates `!` and `@` as the column separators. If the `names`, `places`, and `dates` files are the same as in example 1, then the npd file contains the following:

```
rachel!New York@February 5
jerry!Austin@March 13
mark!Chicago@June 21
linda!Boca Raton@July 16
scott!Seattle@November 4
```

- To list the current directory in four columns, type the following:

```
ls | paste - - - -
```

Each hyphen (`-`) tells the **paste** command to create a column containing data read from the standard input. The first line is put in the first column, the second line in the second column, and so on.

Numbering lines in text files (nl command)

Use the **nl** command to read the specified file (standard input by default), numbers the lines in the input, and writes the numbered lines to standard output.

See the following examples:

- To number only the lines that are not blank, type the following:

```
nl chap1
```

This displays a numbered listing of `chap1`, numbering only the lines that are not blank in the body sections.

- To number all lines, type the following:

```
nl -ba chap1
```

This numbers all the lines in the file named `chap1`, including blank lines.

Removing columns in text files (colrm command)

Use the **colrm** command to remove specified columns from a file. Input is taken from standard input. Output is sent to standard output.

If the command is called with one parameter, the columns of each line from the specified column to the last column are removed. If the command is called with two parameters, the columns from the first specified column to the second specified column are removed.

Note: Column numbering starts with column 1.

See the following examples:

- To remove columns from the `text.fil` file, type the following:

```
colrm 6 < text.fil
```

If `text.fil` contains:

```
123456789
```

then the **colrm** command displays:

```
12345
```

File and directory links

Links are connections between a file name and an index node reference number (i-node number), the internal representation of a file. Because directory entries contain file names paired with i-node numbers, every directory entry is a link.

The i-node number actually identifies the file, not the file name. By using links, any i-node number or file can be known by many different names. For example, i-node number 798 contains a memo regarding June sales in the Omaha office. Presently, the directory entry for this memo is as follows:

i-node Number	File Name
798	**memo**

Because this information relates to information stored in the `sales` and `omaha` directories, linking is used to share the information where it is needed. Using the **ln** command, links are created to these directories. Now the file has three file names as follows:

i-node Number	File Name
798	`memo`
798	`sales/june`
798	`omaha/junesales`

When you use the **pg** or **cat** command to view the contents of any of the three file names, the same information is displayed. If you edit the contents of the i-node number from any of the three file names, the contents of the data displayed by all of the file names will reflect any changes.

Types of links

There are two types of links: hard and symbolic.

Links are created with the **ln** command and are of the following types:

Item	Description
hard link	Allows access to the data of a file from a new file name. Hard links ensure the existence of a file. When the last hard link is removed, the i-node number and its data are deleted. Hard links can be created only between files that are in the same file system.
symbolic link	Allows access to data in other file systems from a new file name. The symbolic link is a special type of file that contains a path name. When a process encounters a symbolic link, the process may search that path. Symbolic links do not protect a file from deletion from the file system.

Note: The user who creates a file retains ownership of that file no matter how many links are created. Only the owner of the file or the root user can set the access mode for that file. However, changes can be made to the file from a linked file name with the proper access mode.

A file or directory exists as long as there is one hard link to the i-node number for that file. In the long listing displayed by the **ls -l** command, the number of hard links to each file and subdirectory is given. All hard links are treated equally by the operating system, regardless of which link was created first.

Linking files (ln command)

Linking files with the **ln** command is a convenient way to work with the same data as if it were in more than one place.

Links are created by giving alternate names to the original file. The use of links allows a large file, such as a database or mailing list, to be shared by several users without making copies of that file. Not only do links save disk space, but changes made to one file are automatically reflected in all the linked files.

The **ln** command links the file designated in the **SourceFile** parameter to the file designated by the **TargetFile** parameter or to the same file name in another directory specified by the **TargetDirectory** parameter. By default, the **ln** command creates hard links. To use the **ln** command to create symbolic links, add the **-s** flag.

Note: You cannot link files across file systems without using the **-s** flag.

If you are linking a file to a new name, you can list only one file. If you are linking to a directory, you can list more than one file.

The **TargetFile** parameter is optional. If you do not designate a target file, the **ln** command creates a file in your current directory. The new file inherits the name of the file designated in the **SourceFile** parameter.

See the following examples:

- To create a link to a file named `chap1`, type the following:

  ```
  ln -f chap1 intro
  ```

  This links `chap1` to the new name, `intro`. When the **-f** flag is used, the file name `intro` is created if it does not already exist. If `intro` does exist, the file is replaced by a link to `chap1`. Both the `chap1` and `intro` file names refer to the same file.
- To link a file named `index` to the same name in another directory named `manual`, type the following:

  ```
  ln index manual
  ```

  This links `index` to the new name, `manual/index`.
- To link several files to names in another directory, type the following:

  ```
  ln chap2 jim/chap3 /home/manual
  ```

  This links `chap2` to the new name `/home/manual/chap2` and `jim/chap3` to `/home/manual/chap3`.
- To use the **ln** command with pattern-matching characters, type the following:

  ```
  ln manual/* .
  ```

  Note: You must type a space between the asterisk and the period.

  This links all files in the `manual` directory into the current directory, dot (.), giving them the same names they have in the `manual` directory.
- To create a symbolic link, type the following:

  ```
  ln -s /tmp/toc toc
  ```

  This creates the symbolic link, **toc**, in the current directory. The `toc` file points to the `/tmp/toc` file. If the `/tmp/toc` file exists, the **cat toc** command lists its contents.
- To achieve identical results without designating the **TargetFile** parameter, type the following:

  ```
  ln -s /tmp/toc
  ```

Command for removing linked files

The **rm** command removes the link from the file name that you indicate.

When one of several hard-linked file names is deleted, the file is not completely deleted because it remains under the other name. When the last link to an i-node number is removed, the data is removed as well. The i-node number is then available for reuse by the system.

DOS files

The AIX operating system allows you to work with DOS files on your system.

Copy to a diskette the DOS files you want to work with. Your system can read these files into a base operating system directory in the correct format and back onto the diskette in DOS format.

Note: The wildcard characters * and ? (asterisk and question mark) do not work correctly with the commands discussed in this section (although they do with the base operating system shell). If you do not specify a file name extension, the file name is matched as if you had specified a blank extension.

Copying DOS files to base operating system files

Use the **dosread** command to copy the specified DOS file to the specified base operating system file.

Note: DOS file-naming conventions are used with one exception. Because the backslash (\) character can have special meaning to the base operating system, use a slash (/) character as the delimiter to specify subdirectory names in a DOS path name.

See the following examples:

- To copy a text file named `chap1.doc` from a DOS diskette to the base operating file system, type the following:

  ```
  dosread -a chap1.doc chap1
  ```

  This copies the DOS text file `\CHAP1.DOC` on the `/dev/fd0` default device to the base operating system file `chap1` in the current directory.
- To copy a binary file from a DOS diskette to the base operating file system, type the following:

  ```
  dosread -D/dev/fd0 /survey/test.dta /home/fran/testdata
  ```

  This copies the `\SURVEY\TEST.DTA` DOS data file on `/dev/fd0` to the base operating system file `/home/fran/testdata`.

Copying base operating system files to DOS files

Use the **doswrite** command to copy the specified base operating system file to the specified DOS file.

Note: DOS file-naming conventions are used with one exception. Because the backslash (\) character can have special meaning to the base operating system, use a slash (/) character as the delimiter to specify subdirectory names in a DOS path name.

See the following examples:

- To copy a text file named `chap1` from the base operating file system to a DOS diskette, type the following:

  ```
  doswrite -a chap1 chap1.doc
  ```

  This copies the base operating system file `chap1` in the current directory to the DOS text file `\CHAP1.DOC` on `/dev/fd0`.
- To copy a binary file named `/survey/test.dta` from the base operating file system to a DOS diskette, type the following:

  ```
  doswrite -D/dev/fd0 /home/fran/testdata /survey/test.dta
  ```

This copies the base operating system data file `/home/fran/testdata` to the DOS file `\SURVEY\TEST.DTA` on `/dev/fd0`.

Deleting DOS files

Use the **dosdel** command to delete the specified DOS file.

Note: DOS file-naming conventions are used with one exception. Because the backslash (\) character can have special meaning to the base operating system, use a slash (/) character as the delimiter to specify subdirectory names in a DOS path name.

The **dosdel** command converts lowercase characters in the file or directory name to uppercase before it checks the disk. Because all file names are assumed to be full (not relative) path names, you need not add the initial slash (/).

For example, to delete a DOS file named `file.ext` on the default device (`/dev/fd0`), type the following:

```
dosdel file.ext
```

Displaying contents of a DOS directory

Use the **dosdir** command to display information about the specified DOS files or directories.

Note: DOS file-naming conventions are used with one exception. Because the backslash (\) character can have special meaning to the base operating system, use a slash (/) character as the delimiter to specify subdirectory names in a DOS path name.

The **dosdir** command converts lowercase characters in the file or directory name to uppercase before it checks the disk. Because all file names are assumed to be full (not relative) path names, you need not add the initial / (slash).

For example, to read a directory of the DOS files on `/dev/fd0`, type the following:

```
dosdir
```

The command returns the names of the files and disk-space information, similar to the following.

```
PG3-25.TXT
PG4-25.TXT
PG5-25.TXT
PG6-25.TXT
Free space: 312320 bytes
```

Command summary for files

The following are commands for files, file handling procedures, and DOS files. There is also a list of commands for linking files and directories.

Table 59. Commands for files

Item	Description
*****	Wildcard, matches any characters
?	Wildcard, matches any single character
[ ]	Metacharacters, matches enclosed characters.

Table 60. Commands for file handling procedures

Item	Description
cat	Concatenates or displays files
cmp	Compares two files
colrm	Extracts columns from a file

Table 60. Commands for file handling procedures (continued)

Item	Description
cp	Copies files
cut	Writes out selected bytes, characters, or fields from each line of a file
diff	Compares text files
file	Determines the file type
find	Finds files with a matching expression
grep	Searches a file for a pattern
head	Displays the first few lines or bytes of a file or files
more	Displays continuous text one screen at a time on a display screen
mv	Moves files
nl	Numbers lines in a file
pg	Formats files to the display
rm	Removes (unlinks) files or directories
paste	Merges the lines of several files or subsequent lines in one file
sort	Sorts files, merges files that are already sorted, and checks files to determine if they have been sorted
tail	Writes a file to standard output, beginning at a specified point
wc	Counts the number of lines, words, and bytes in a file

Table 61. Command for linking files and directories

Item	Description
ln	Links files and directories

Table 62. Commands for DOS files

Item	Description
dosdel	Deletes DOS files
dosdir	Lists the directory for DOS files
dosread	Copies DOS files to Base Operating System files
doswrite	Copies Base Operating System files to DOS files

Operating system shells

Your interface to the operating system is called a *shell*.

The shell is the outermost layer of the operating system. Shells incorporate a programming language to control processes and files, as well as to start and control other programs. The shell manages the interaction between you and the operating system by prompting you for input, interpreting that input for the operating system, and then handling any resulting output from the operating system.

Shells provide a way for you to communicate with the operating system. This communication is carried out either interactively (input from the keyboard is acted upon immediately) or as a shell script. A *shell script* is a sequence of shell and operating system commands that is stored in a file.

When you log in to the system, the system locates the name of a shell program to execute. After it is executed, the shell displays a command prompt. This prompt is usually a $ (dollar sign). When you type a

command at the prompt and press the Enter key, the shell evaluates the command and attempts to carry it out. Depending on your command instructions, the shell writes the command output to the screen or redirects the output. It then returns the command prompt and waits for you to type another command.

A *command line* is the line on which you type. It contains the shell prompt. The basic format for each line is as follows:

```
$ Command Argument(s)
```

The shell considers the first word of a command line (up to the first blank space) as the command and all subsequent words as arguments.

Note: When `libc.a` is moved or renamed, the `Killed` error message is displayed from the shell because there is no `libc.a` file available for the system to load and run the utilities. The **recsh** command invokes the recovery shell, which provides an ability to rename `libc.a` if it is accidently moved.

Related tasks

Listing previously entered commands (history command)
Use the **history** command to list commands that you have previously entered.

Shell concepts

Before you start working with the different types of shells available for AIX you need to understand basic terminology and features.

Available shells

The following are the shells that are provided with AIX.

- Korn shell (started with the **ksh** command)
- Bourne shell (started with the **bsh** command)
- Restricted shell (a limited version of the Bourne shell, and started with the **Rsh** command)
- POSIX shell (also known as the Korn Shell, and started with the **psh** command)
- Restricted shell for the Korn shell (**ksh** and **ksh93**). The **ksh** and **ksh93** shells are provided with their restricted shell equivalents **rksh** and **rksh93**.
- Default shell (started with the **sh** command)
- C shell (started with the **csh** command)
- Trusted shell (a limited version of the Korn shell, and started with the **tsh** command)
- Remote shell (started with the **rsh** command)

The *login shell* refers to the shell that is loaded when you log in to the computer system. Your login shell is set in the `/etc/passwd` file. The Korn shell is the standard operating system login shell and is backward-compatible with the Bourne Shell.

The Korn shell (`/usr/bin/ksh`) is set up as the default shell. The default or standard shell refers to the shells linked to and started with the `/usr/bin/sh` command. The Bourne shell (`/usr/bin/sh`) can be substituted as the default shell. The POSIX shell, which is invoked by the `/usr/bin/psh` command, resides as a link to the `/usr/bin/sh` command.

Related concepts

Bourne shell
The Bourne shell is an interactive command interpreter and command programming language.

Korn shell or POSIX shell commands

The Korn shell is an interactive command interpreter and command programming language. It conforms to the Portable Operating System Interface for Computer Environments (POSIX), an international standard for operating systems.

Shells terminology

The terms and definitions in this table are helpful in understanding shells.

Item	Description
blank	A blank is one of the characters in the blank character class defined in the **LC_CTYPE** category. In the POSIX shell, a blank is either a tab or space.
built-in command	A command that the shell executes without searching for it and creating a separate process.
command	A sequence of characters in the syntax of the shell language. The shell reads each command and carries out the desired action either directly or by invoking separate utilities.
comment	Any word that begins with pound sign (#). The word and all characters that follow it, until the next newline character, are ignored.
identifier	A sequence of letters, digits, or underscores from the portable character set, starting with a letter or underscore. The first character of an identifier must not be a digit. Identifiers are used as names for aliases, functions, and named parameters.
list	A sequence of one or more pipelines separated by one of the following symbols: semicolon (;), ampersand (&), double ampersand (&&), or double bar (||). The list is optionally ended by one of the following symbols: semicolon (;), ampersand (&), or bar ampersand (|&). **;** Sequentially processes the preceding pipeline. The shell carries out each command in turn and waits for the most recent command to complete. **&** Asynchronously processes the preceding pipeline. The shell carries out each command in turn, processing the pipeline in the background without waiting for it to complete. **|&** Asynchronously processes the preceding pipeline and establishes a two-way pipe to the parent shell. The shell carries out each command in turn, processing the pipeline in the background without waiting for it to complete. The parent shell can read from and write to the standard input and output of the created command by using the **read -p** and **print -p** commands. Only one such command can be active at any given time. **&&** Processes the list that follows this symbol only if the preceding pipeline returns an exit value of zero (0). **||** Processes the list that follows this symbol only if the preceding pipeline returns a nonzero exit value. The semicolon (;), ampersand (&), and bar ampersand (|&) have a lower priority than the double ampersand (&&) and double bar (||). The ;, &, and |& symbols have equal priority among themselves. The && and || symbols are equal in priority. One or more newline characters can be used instead of a semicolon to delimit two commands in a list. **Note:** The |& symbol is valid only in the Korn shell.

Item	Description
metacharacter	Each metacharacter has a special meaning to the shell and causes termination of a word unless it is quoted. Metacharacters are: pipe (`|`), ampersand (&), semicolon (`;`), less-than sign (<), greater-than sign (>), left parenthesis (`(`), right parenthesis (`)`), dollar sign ($), backquote (`` ` ``), backslash (\), right quote (`'`), double quotation marks (`"`), newline character, space character, and tab character. All characters enclosed between single quotation marks are considered quoted and are interpreted literally by the shell. The special meaning of metacharacters is retained if not quoted. (Metacharacters are also known as *parser metacharacters* in the C shell.)
parameter assignment list	Includes one or more words of the form *Identifier=Value* in which spaces surrounding the equal sign (=) must be balanced. That is, leading and trailing blanks, or no blanks, must be used. **Note:** In the C shell, the parameter assignment list is of the form **set***Identifier=Value*. The spaces surrounding the equal sign (=) are required.
pipeline	A sequence of one or more commands separated by pipe (`|`). Each command in the pipeline, except possibly the last command, is run as a separate process. However, the standard output of each command that is connected by a pipe becomes the standard input of the next command in the sequence. If a list is enclosed with parentheses, it is carried out as a simple command that operates in a separate subshell. If the reserved word `!` does not precede the pipeline, the exit status will be the exit status of the last command specified in the pipeline. Otherwise, the exit status is the logical NOT of the exit status of the last command. In other words, if the last command returns zero, the exit status will be 1. If the last command returns greater than zero, the exit status will be zero. The format for a pipeline is as follows: `[!] command1 [	command2 ...]` **Note:** Early versions of the Bourne shell used the caret (^) to indicate a pipe.
shell variable	A name or parameter to which a value is assigned. Assign a variable by typing the variable name, an equal sign (=), and then the value. The variable name can be substituted for the assigned value by preceding the variable name with a dollar sign ($). Variables are particularly useful for creating a short notation for a long path name, such as $HOME for the home directory. A predefined variable is one whose value is assigned by the shell. A user-defined variable is one whose value is assigned by a user.
simple command	A sequence of optional parameter assignment lists and redirections, in any sequence. They are optionally followed by commands, words, and redirections. They are terminated by `;`, `|`, &, `||`, &&, `|&`, or a newline character. The command name is passed as parameter `0` (as defined by the `exec` subroutine). The value of a simple command is its exit status of zero if it terminates normally or nonzero if it terminates abnormally. The sigaction, sigvec, or signal subroutine includes a list of signal-exit status values.
subshell	A shell that is running as a child of the login shell or the current shell.
wildcard character	Also known as a *pattern-matching character*. The shell associates them with assigned values. The basic wildcards are `?`, `*`, `[set]`, and `[!set]`. Wildcard characters are particularly useful when performing file name substitution.
word	A sequence of characters that does not contain any blanks. Words are separated by one or more metacharacters.

Specifying a shell for a script file

When you run an executable shell script in either the Korn (the POSIX Shell) or Bourne shell, the commands in the script are carried out under the control of the current shell (the shell from which the script is started) unless you specify a different shell. When you run an executable shell script in the C shell, the commands in the script are carried out under the control of the Bourne shell (`/usr/bin/bsh`) unless you specify a different shell.

You can run a shell script in a specific shell by including the shell within the shell script.

To run an executable shell script under a specific shell, type `#!`*`Path`* on the first line of the shell script, and press Enter. The `#!` characters identify the file type. The *Path* variable specifies the path name of the shell from which to run the shell script.

For example, to run the **bsh** script in the Bourne shell, type the following:

```
#!/usr/bin/bsh
```

When you precede a shell script file name with a shell command, the shell specified on the command line overrides any shell specified within the script file itself. Therefore, typing `ksh myfile` and pressing Enter runs the file named `myfile` under the control of the Korn shell, even if the first line of `myfile` is `#!/usr/bin/csh`.

Shell features

There are advantages to using the shell as an interface to the system.

The primary advantages of interfacing to the system through a shell are as follows:

- **Wildcard substitution in file names (pattern-matching)**

  Carries out commands on a group of files by specifying a pattern to match, rather than specifying an actual file name.

  For more information, see:

  – “File name substitution in the Korn shell or POSIX shell” on page 227
  – “File name substitution in the Bourne shell” on page 260
  – “File name substitution in the C shell” on page 279

- **Background processing**

  Sets up lengthy tasks to run in the background, freeing the terminal for concurrent interactive processing.

  For more information, see the **bg** command in the following:

  – “Job control in the Korn shell or POSIX shell” on page 243
  – “C shell built-in commands” on page 286

  Note: The Bourne shell does not support job control.

- **Command aliasing**

  Gives an alias name to a command or phrase. When the shell encounters an alias on the command line or in a shell script, it substitutes the text to which the alias refers.

  For more information, see:

  – “Command aliasing in the Korn shell or POSIX shell” on page 256
  – “Alias substitution in the C shell” on page 276

  Note: The Bourne shell does not support command aliasing.

- **Command history**

  Records the commands you enter in a history file. You can use this file to easily access, modify, and reissue any listed command.

For more information, see the **history** command in the following:

- "Korn shell or POSIX shell command history" on page 256
- "C shell built-in commands" on page 286
- "History substitution in the C shell" on page 297

Note: The Bourne shell does not support command history.

- **File name substitution**

  Automatically produces a list of file names on a command line using pattern-matching characters.

  For more information, see:

  - "File name substitution in the Korn shell or POSIX shell" on page 227
  - "File name substitution in the Bourne shell" on page 260
  - "File name substitution in the C shell" on page 279

- **Input and output redirection**

  Redirects input away from the keyboard and redirects output to a file or device other than the terminal. For example, input to a program can be provided from a file and redirected to the printer or to another file.

  For more information, see:

  - "Input and output redirection in the Korn shell or POSIX shell" on page 228
  - "Input and output redirection in the Bourne shell" on page 261
  - "Input and output redirection in the C shell" on page 299

- **Piping**

  Links any number of commands together to form a complex program. The standard output of one program becomes the standard input of the next.

  For more information, see the *pipeline* definition in "Shells terminology" on page 201.

- **Shell variable substitution**

  Stores data in user-defined variables and predefined shell variables.

  For more information, see:

  - "Parameter substitution in the Korn shell or POSIX shell" on page 225
  - "Variable substitution in the Bourne shell" on page 271
  - "Variable substitution in the C shell" on page 277

Related concepts

Commands

Some commands can be entered simply by typing one word. It is also possible to combine commands so that the output from one command becomes the input for another command.

Character classes

You can use character classes to match file names.

You can use character classes to match file names, as follows:

```
[[:charclass:]]
```

This format instructs the system to match any single character belonging to the specified class. The defined classes correspond to `ctype` subroutines, as follows:

Character Class	Definition
alnum	Alphanumeric characters

Character Class	Definition
alpha	Uppercase and lowercase letters
blank	Space or horizontal tab
cntrl	Control characters
digit	Digits
graph	Graphic characters
lower	Lowercase letters
print	Printable characters
punct	Punctuation characters
space	Space, horizontal tab, carriage return, newline, vertical tab, or form-feed character
upper	Uppercase characters
xdigit	Hexadecimal digits

Restricted shell

The restricted shell is used to set up login names and execution environments whose capabilities are more controlled than those of the regular Bourne shell.

The **Rsh** or **bsh -r** command opens the restricted shell. The behavior of these commands is identical to those of the **bsh** command, except that the following actions are not allowed:

- Changing the directory (with the **cd** command)
- Setting the value of *PATH* or *SHELL* variables
- Specifying path or command names containing a slash (/)
- Redirecting output

If the restricted shell determines that a command to be run is a shell procedure, it uses the Bourne shell to run the command. In this way, it is possible to provide a user with shell procedures that access the full power of the Bourne shell while imposing a limited menu of commands. This situation assumes that the user does not have write and execute permissions in the same directory.

If the *File* [*Parameter*] parameter is specified when the Bourne shell is started, the shell runs the script file identified by the *File* parameter, including any parameters specified. The script file specified must have read permission. Any **setuid** and **setgid** settings for script files are ignored. The shell then reads the commands. If either the **-c** or **-s** flag is used, do not specify a script file.

When started with the **Rsh** command, the shell enforces restrictions after interpreting the `.profile` and `/etc/environment` files. Therefore, the writer of the `.profile` file has complete control over user actions by performing setup actions and leaving the user in an appropriate directory (probably not the login directory). An administrator can create a directory of commands in the `/usr/rbin` directory that the **Rsh** command can use by changing the *PATH* variable to contain the directory. If it is started with the **bsh -r** command, the shell applies restrictions when interpreting the `.profile` files.

When called with the name **Rsh**, the restricted shell reads the user's `.profile` file (`$HOME/.profile`). It acts as the regular Bourne shell while doing this, except that an interrupt causes an immediate exit instead of a return to command level.

The Korn shell can be started as a restricted shell with the command **ksh -r**.

The inodes for **ksh** and **rksh** are identical, and the inodes for **ksh93** and **rksh93** are identical.

Creating and running a shell script

A *shell script* is a file that contains one or more commands. Shell scripts provide an easy way to carry out tedious commands, large or complicated sequences of commands, and routine tasks. When you enter the name of a shell script file, the system executes the command sequence contained by the file.

You can create a shell script by using a text editor. Your script can contain both operating system commands and shell built-in commands.

The following steps are general guidelines for writing shell scripts:

1. Using a text editor, create and save a file. You can include any combination of shell and operating system commands in the shell script file. By convention, shell scripts that are not set up for use by many users are stored in the `$HOME/bin` directory.

   Note: The operating system does not support the `setuid` or `setgid` subroutines within a shell script.
2. Use the **chmod** command to allow only the owner to run (or execute) the file. For example, if your file is named `script1`, type the following:

   ```
   chmod u=rwx script1
   ```
3. Type the script name on the command line to run the shell script. To run the `script1` shell script, type the following:

   ```
   script1
   ```

   Note: You can run a shell script without making it executable if a shell command (**ksh**, **bsh**, or **csh**) precedes the shell script file name on the command line. For example, to run a nonexecutable file named `script1` under the control of the Korn shell, type the following:

   ```
   ksh script1
   ```

Related concepts

Commands

Some commands can be entered simply by typing one word. It is also possible to combine commands so that the output from one command becomes the input for another command.

Korn shell

The Korn shell (ksh command) is backwardly compatible with the Bourne shell (bsh command) and contains most of the Bourne shell features as well as several of the best features of the C shell.

Variables set by the Korn shell or POSIX shell

The following are variables that are set by the shell.

Item	Description
underscore (_)	Indicates initially the absolute path name of the shell or script being executed as passed in the environment. Subsequently, it is assigned the last argument of the previous command. This parameter is not set for commands that are asynchronous. This parameter is also used to hold the name of the matching `MAIL` file when checking for mail.
ERRNO	Specifies a value that is set by the most recently failed subroutine. This value is system-dependent and is intended for debugging purposes.
LINENO	Specifies the line number of the current line within the script or function being executed.
OLDPWD	Indicates the previous working directory set by the **cd** command.
OPTARG	Specifies the value of the last option argument processed by the **getopts** regular built-in command.

Item	Description
OPTIND	Specifies index of the last option argument processed by the **getopts** regular built-in command.
PPID	Identifies the process number of the parent of the shell.
PWD	Indicates the present working directory set by the **cd** command.
RANDOM	Generates a random integer, uniformly distributed between 0 and 32767. The sequence of random numbers can be initialized by assigning a numeric value to the *RANDOM* variable.
REPLY	Set by the **select** statement and by the **read** regular built-in command when no arguments are supplied.
SECONDS	Specifies the number of seconds since shell invocation is returned. If this variable is assigned a value, then the value returned upon reference will be the value that was assigned plus the number of seconds since the assignment.

Variables used by the Korn shell or POSIX shell

The following are variables that are used by the shell.

Item	Description
CDPATH	Indicates the search path for the **cd** (change directory) command.
COLUMNS	Defines the width of the edit window for the shell edit modes and for printing **select** lists.
EDITOR	If the value of this parameter ends in emacs, gmacs, or vi, and the *VISUAL* variable is not set with the **set** special built-in command, then the corresponding option is turned on.
ENV	If this variable is set, then parameter substitution is performed on the value to generate the path name of the script that will be executed when the shell is invoked. This file is typically used for alias and function definitions. This variable will be ignored for noninteractive shells.
FCEDIT	Specifies the default editor name for the **fc** regular built-in command.
FPATH	Specifies the search path for function definitions. This path is searched when a function with the **-u** flag is referenced and when a command is not found. If an executable file is found, then it is read and executed in the current environment.
HISTFILE	If this variable is set when the shell is invoked, then the value is the path name of the file that will be used to store the command history. The initialization process for the `history` file can be dependent on the system start-up files because some start-up files can contain commands that effectively preempt the settings the user has specified for HISTFILE and HISTSIZE. For example, function definition commands are recorded in the `history` file. If the system administrator includes function definitions in a system start-up file that is called before the ENV file or before *HISTFILE* or *HISTSIZE* variable is set, the `history` file is initialized before the user can influence its characteristics.
HISTSIZE	If this variable is set when the shell is invoked, then the number of previously entered commands that are accessible by this shell will be greater than or equal to this number. The default is 128 commands for nonroot users and 512 commands for the root user.

Item	Description
HOME	Indicates the name of your login directory, which becomes the current directory upon completion of a login. The **login** program initializes this variable. The **cd** command uses the value of the *$HOME* parameter as its default value. Using this variable rather than an explicit path name in a shell procedure allows the procedure to be run from a different directory without alterations.
IFS	Specifies IFS (internal field separators), which are normally space, tab, and newline, used to separate command words that result from command or parameter substitution and for separating words with the regular built-in command **read**. The first character of the *IFS* parameter is used to separate arguments for the $* substitution.

KSH_CHOWN_MB_SUPPORT	If the language (*LANG*) variable is changed inside the Korn shell (ksh) script, by default the ksh script ignores the *LC_CTYPE* variable. Therefore, the wildcard (*) expansion fails for few locales.
	To solve this issue, you can export the *KSH_CHOWN_MB_SUPPORT* environment variable by using the following command:
	`export KSH_CHOWN_MB_SUPPORT=ON`
	You can turn off the *KSH_CHOWN_MB_SUPPORT* environment variable by using the following command:
	`unset KSH_CHOWN_MB_SUPPORT`

KSH_STAK_SIZE	Increases the internal stack size of the Korn shell (ksh). By default, the size of the internal stack is 16 KB. To increase the size of the internal stack to process large files, use the *KSH_STAK_SIZE* environment variable.
	The values of the *KSH_STAK_SIZE* environment variable must be in the range 32 KB - 64 MB. The following examples show the format that you must use to specify the *KSH_STAK_SIZE* environment variable:
	• To increase the size of the internal stack to 1 MB, type the following command:
	`export KSH_STAK_SIZE=1MB`
	• To increase the size of the internal stack to 30 KB, type the following command:
	`export KSH_STAK_SIZE=30KB`
	Although you want to increase the internal stack size to 30 KB, the *KSH_STAK_SIZE* environment variable rounds off the value to 32 KB because the *KSH_STAK_SIZE* environment variable must be in the range 32 KB - 64 MB.
	Similarly, if you want to increase the internal stack size to a value greater than 64 MB, the *KSH_STAK_SIZE* environment variable still rounds off the value to 32 KB.

LANG	Provides a default value for the *LC_** variables.
LC_ALL	Overrides the value of the *LANG* and *LC_** variables.
LC_COLLATE	Determines the behavior of range expression within pattern matching.
LC_CTYPE	Defines character classification, case conversion, and other character attributes.
LC_MESSAGES	Determines the language in which messages are written.
LINES	Determines the column length for printing select lists. Select lists print vertically until about two-thirds of lines specified by the *LINES* variable are filled.

Item	Description
MAIL	Specifies the file path name used by the mail system to detect the arrival of new mail. If this variable is set to the name of a mail file and the *MAILPATH* variable is not set, then the shell informs the user of new mail in the specified file.
MAILCHECK	Specifies how often (in seconds) the shell checks for changes in the modification time of any of the files specified by the *MAILPATH* or *MAIL* variables. The default value is 600 seconds. When the time has elapsed, the shell checks before issuing the next prompt.
MAILPATH	Specifies a list of file names separated by colons. If this variable is set, then the shell informs the user of any modifications to the specified files that have occurred during the period, in seconds, specified by the *MAILCHECK* variable. Each file name can be followed by a **?** and a message that will be printed. The message will undergo variable substitution with the *$_* variable defined as the name of the file that has changed. The default message is `you have mail in $_`.
NLSPATH	Determines the location of message catalogs for the processing of **LC_MESSAGES**.
PATH	Indicates the search path for commands, which is an ordered list of directory path names separated by colons. The shell searches these directories in the specified order when it looks for commands. A null string anywhere in the list represents the current directory.
PS1	Specifies the string to be used as the primary system prompt. The value of this parameter is expanded for parameter substitution to define the primary prompt string, which is a $ by default. The `!` character in the primary prompt string is replaced by the command number.
PS2	Specifies the value of the secondary prompt string, which is a > by default.
PS3	Specifies the value of the selection prompt string used within a **select** loop, which is *#?* by default.
PS4	The value of this variable is expanded for parameter substitution and precedes each line of an execution trace. If omitted, the execution trace prompt is a +.
SHELL	Specifies the path name of the shell, which is kept in the environment.
SHELL PROMPT	When used interactively, the shell prompts with the value of the *PS1* parameter before reading a command. If at any time a new line is entered and the shell requires further input to complete a command, the shell issues the secondary prompt (the value of the *PS2* parameter).
TMOUT	Specifies the number of seconds a shell waits inactive before exiting. If the *TMOUT* variable is set to a value greater than zero (0), the shell exits if a command is not entered within the prescribed number of seconds after issuing the *PS1* prompt. (Note that the shell can be compiled with a maximum boundary that cannot be exceeded for this value.) **Note:** After the timeout period has expired, there is a 60-second pause before the shell exits.
VISUAL	If the value of this variable ends in emacs, gmacs, or vi, then the corresponding option is turned on.

The shell gives default values to the *PATH*, *PS1*, *PS2*, *MAILCHECK*, *TMOUT*, and *IFS* parameters, but the *HOME*, *SHELL*, *ENV*, and *MAIL* parameters are *not* set by the shell (although the *HOME* parameter is set by the **login** command).

Command substitution in the Korn shell or POSIX shell

The Korn Shell, or POSIX Shell, lets you perform command substitution. In command substitution, the shell executes a specified command in a subshell environment and replaces that command with its output.

To execute command substitution in the Korn shell or POSIX shell, type the following:

```
$(command)
```

or, for the backquoted version, type the following:

```
`command`
```

Note: Although the backquote syntax is accepted by **ksh**, it is considered obsolete by the X/Open Portability Guide Issue 4 and POSIX standards. These standards recommend that portable applications use the `$(command)` syntax.

The shell expands the command substitution by executing `command` in a subshell environment and replacing the command substitution (the text of `command` plus the enclosing `$( )` or backquotes) with the standard output of the command, removing sequences of one or more newline characters at the end of the substitution.

In the following example, the `$( )` surrounding the command indicates that the output of the **whoami** command is substituted:

```
echo My name is: $(whoami)
```

You can perform the same command substitution with:

```
echo My name is: `whoami`
```

The output from both examples for user dee is:

```
My name is: dee
```

You can also substitute arithmetic expressions by enclosing them in `( )`. For example, the command:

```
echo Each hour contains $((60 * 60)) seconds
```

produces the following result:

```
Each hour contains 3600 seconds
```

The Korn shell or POSIX shell removes all trailing newline characters when performing command substitution. For example, if your current directory contains the `file1`, `file2`, and `file3` files, the command:

```
echo $(ls)
```

removes the newline characters and produces the following output:

```
file1 file2 file3
```

To preserve newline characters, insert the substituted command in `" "`:

```
echo "$(ls)"
```

Arithmetic evaluation in the Korn shell or POSIX shell

The Korn shell or POSIX shell regular built-in **let** command enables you to perform integer arithmetic.

Constants are of the form [**Base**]**Number**. The **Base** parameter is a decimal number between 2 and 36 inclusive, representing the arithmetic base. The **Number** parameter is a number in that base. If you omit the **Base** parameter, the shell uses a base of 10.

Arithmetic expressions use the same syntax, precedence, and associativity of expression as the C programming language. All of the integral operators, other than double plus (++), double hyphen (–), question mark-colon (?:), and comma (,), are supported. The following table lists valid Korn shell or POSIX shell operators in decreasing order of precedence:

Operator	Definition
-	Unary minus
!	Logical negation
~	Bitwise negation
*	Multiplication
/	Division
%	Remainder
+	Addition
-	Subtraction
<<, >>	Left shift, right shift
<=,>=, <>, ==, !=	Comparison
&	Bitwise AND
^	Bitwise exclusive OR
|	Bitwise OR
&&	Logical AND
||	Logical OR
= *=, /=, &= +=, -=, <<=, > >=, &=, ^=, |=	Assignment

Many arithmetic operators, such as *, &, <, and >, have special meaning to the Korn shell or POSIX shell. These characters must be quoted. For example, to multiply the current value of *y* by 5 and reassign the new value to *y*, use the expression:

```
let "y = y * 5"
```

Enclosing the expression in quotation marks removes the special meaning of the * character.

You can group operations inside **let** command expressions to force grouping. For example, in the expression:

```
let "z = q * (z - 10)"
```

the command multiplies *q* by the reduced value of *z*.

The Korn shell or POSIX shell includes an alternative form of the **let** command if only a single expression is to be evaluated. The shell treats commands enclosed in (( )) as quoted expressions. Therefore, the expression:

```
((x = x / 3))
```

is equivalent to:

```
let "x = x / 3"
```

Named parameters are referenced by name within an arithmetic expression without using the parameter substitution syntax. When a named parameter is referenced, its value is evaluated as an arithmetic expression.

Specify an internal integer representation of a named parameter with the **-i** flag of the **typeset** special built-in command. Using the **-i** flag, arithmetic evaluation is performed on the value of each assignment to a named parameter. If you do not specify an arithmetic base, the first assignment to the parameter determines the arithmetic base. This base is used when parameter substitution occurs.

Related concepts

Korn shell or POSIX shell commands
The Korn shell is an interactive command interpreter and command programming language. It conforms to the Portable Operating System Interface for Computer Environments (POSIX), an international standard for operating systems.

Parameters in the Korn shell
Korn shell parameters are discussed below.

Field splitting in the Korn shell or POSIX shell

After performing command substitution, the Korn shell scans the results of substitutions for those field separator characters found in the **IFS** (Internal Field Separator) variable. Where such characters are found, the shell splits the substitutions into distinct arguments.

The shell retains explicit null arguments (`""` or `''`) and removes implicit null arguments (those resulting from parameters that have no values).

- If the value of *IFS* is a space, tab, or newline character, or if it is not set, any sequence of space, tab, or newline characters at the beginning or end of the input will be ignored and any sequence of those characters within the input will delimit a field. For example, the following input yields two fields, **school** and **days**:

  ```
  <newline><space><tab>school<tab><tab>days<space>
  ```

- Otherwise, and if the value of **IFS** is not null, the following rules apply in sequence. ***IFS*** *white space* is used to mean any sequence (zero or more instances) of white-space characters that are in the **IFS** value (for example, if **IFS** contains space/comma/tab, any sequence of space and tab characters is considered **IFS** white space).

  1. **IFS** white space is ignored at the beginning and end of the input.
  2. Each occurrence in the input of an **IFS** character that is not **IFS** white space, along with any adjacent **IFS** white space, delimits a field.
  3. Nonzero length **IFS** white space delimits a field.

List of Korn shell or POSIX shell special built-in commands

Special commands are built into the Korn shell and POSIX shell and executed in the shell process.

Item	Description
: (colon)	Expands only arguments.
. (dot)	Reads a specified file and then executes the commands.
break	Exits from the enclosing **for**, **while**, **until**, or **select** loop, if one exists.
continue	Resumes the next iteration of the enclosing **for**, **while**, **until**, or **select** loop.
eval	Reads the arguments as input to the shell and executes the resulting command or commands.
exec	Executes the command specified by the *Argument* parameter, instead of this shell, without creating a new process.
exit	Exits the shell whose exit status is specified by the *n* parameter.
export	Marks names for automatic export to the environment of subsequently executed commands.
newgrp	Equivalent to the `exec/usr/bin/newgrp` [*Group* ...] command.

Item	Description
readonly	Marks the specified names read-only.
return	Causes a shell to return to the invoking script.
set	Unless options or arguments are specified, writes the names and values of all shell variables in the collation sequence of the current locale.
shift	Renames positional parameters.
times	Prints the accumulated user and system times for both the shell and the processes run from the shell.
trap	Runs a specified command when the shell receives a specified signal or signals.
typeset	Sets attributes and values for shell parameters.
unset	Unsets the values and attributes of the specified parameters.

Related concepts
Korn shell or POSIX shell built-in commands
Special commands are built in to the Korn shell and POSIX shell and executed in the shell process.

Korn shell or POSIX shell regular built-in commands
The following is a list of the Korn shell or POSIX shell regular built-in commands.

Item	Description
alias	Prints a list of aliases to standard output.
bg	Puts specified jobs in the background.
cd	Changes the current directory to the specified directory or substitutes the current string with the specified string.
echo	Writes character strings to standard output.
fc	Selects a range of commands from the last *HISTSIZE* variable command typed at the terminal. Re-executes the specified command after old-to-new substitution is performed.
fg	Brings the specified job to the foreground.
getopts	Checks the *Argument* parameter for legal options.
jobs	Lists information for the specified jobs.
kill	Sends the **TERM** (terminate) signal to specified jobs or processes.
let	Evaluates specified arithmetic expressions.
print	Prints shell output.
pwd	Equivalent to the **print -r -$PWD** command.
read	Takes shell input.
ulimit	Sets or displays user process resource limits as defined in the `/etc/security/limits` file.
umask	Determines file permissions.
unalias	Removes the parameters in the list of names from the alias list.
wait	Waits for the specified job and terminates.
whence	Indicates how each specified name would be interpreted if used as a command name.

For more information, see "Korn shell or POSIX shell built-in commands" on page 231.

Related concepts

Korn shell or POSIX shell built-in commands

Special commands are built in to the Korn shell and POSIX shell and executed in the shell process.

Conditional expressions for the Korn shell or POSIX shell

A conditional expression is used with the `[[` compound command to test attributes of files and to compare strings.

Word splitting and file name substitution are not performed on words appearing between `[[` and `]]`. Each expression is constructed from one or more of the following unary or binary expressions:

Item	Description
-a *File*	True, if the specified file is a symbolic link that points to another file that does exist.
-b *File*	True, if the specified file exists and is a block special file.
-c *File*	True, if the specified file exists and is a character special file.
-d *File*	True, if the specified file exists and is a directory.
-e *File*	True, if the specified file exists.
-f *File*	True, if the specified file exists and is an ordinary file.
-g *File*	True, if the specified file exists and its **setgid** bit is set.
-h *File*	True, if the specified file exists and is a symbolic link.
-k *File*	True, if the specified file exists and its sticky bit is set.
-n *String*	True, if the length of the specified string is nonzero.
-o *Option*	True, if the specified option is on.
-p *File*	True, if the specified file exists and is a FIFO special file or a pipe.
-r *File*	True, if the specified file exists and is readable by the current process.
-s *File*	True, if the specified file exists and has a size greater than 0.
-t *FileDescriptor*	True, if specified file descriptor number is open and associated with a terminal device.
-u *File*	True, if the specified file exists and its **setuid** bit is set.
-w *File*	True, if the specified file exists and the write bit is on. However, the file will not be writable on a read-only file system even if this test indicates true.
-x *File*	True, if the specified file exists and the **execute** flag is on. If the specified file exists and is a directory, then the current process has permission to search in the directory.
-z *String*	True, if length of the specified string is 0.
-L *File*	True, if the specified file exists and is a symbolic link.
-O *File*	True, if the specified file exists and is owned by the effective user ID of this process.
-G *File*	True, if the specified file exists and its group matches the effective group ID of this process.
-S *File*	True, if the specified file exists and is a socket.
File1 **-nt** *File2*	True, if *File1* exists and is newer than *File2*.
File1 **-ot** *File2*	True, if *File1* exists and is older than *File2*.
File1 **-ef** *File2*	True, if *File1* and *File2* exist and refer to the same file.

Item	Description
String1 **=** *String2*	True, if *String1* is equal to *String2*.
String1 **!=** *String2*	True, if *String1* is not equal to *String2*.
String **=** *Pattern*	True, if the specified string matches the specified pattern.
String **!=** *Pattern*	True, if the specified string does not match the specified pattern.
String1 **<** *String2*	True, if *String1* comes before *String2* based on the ASCII value of their characters.
String1 **>** *String2*	True, if *String1* comes after *String2* based on the ASCII value of their characters.
Expression1 **-eq** *Expression2*	True, if *Expression1* is equal to *Expression2*.
Expression1 **-ne** *Expression2*	True, if *Expression1* is not equal to *Expression2*.
Expression1 **-lt** *Expression2*	True, if *Expression1* is less than *Expression2*.
Expression1 **-gt** *Expression2*	True, if *Expression1* is greater than *Expression2*.
Expression1 **-le** *Expression2*	True, if *Expression1* is less than or equal to *Expression2*.
Expression1 **-ge** *Expression2*	True, if *Expression1* is greater than or equal to *Expression2*.

Note: In each of the previous expressions, if the *File* variable is similar to `/dev/fd/`*n*, where *n* is an integer, then the test is applied to the open file whose descriptor number is *n*.

You can construct a compound expression from these primitives, or smaller parts, by using any of the following expressions, listed in decreasing order of precedence:

Item	Description
(*Expression***)**	True, if the specified expression is true. Used to group expressions.
! *Expression*	True, if the specified expression is false.
Expression1 **&&** *Expression2*	True, if *Expression1* and *Expression2* are both true.
Expression1 **||** *Expression2*	True, if either *Expression1* or *Expression2* is true.

Quotation of characters in the Korn shell or POSIX shell

When you want the Korn shell or POSIX shell to read a character as a regular character, rather than with any normally associated meaning, you must *quote* it.

Each metacharacter has a special meaning to the shell and, unless quoted, causes termination of a word. The following characters are considered metacharacters by the Korn shell or POSIX shell and must be quoted if they are to represent themselves:

- pipe (`|`)
- ampersand (&)
- semicolon (`;`)
- less-than sign (<) and greater-than sign (>)
- left parenthesis (`(`) and right parenthesis (`)`)
- dollar sign ($)
- backquote (`` ` ``) and single quotation mark (`'`)

- backslash (\)
- double-quotation marks (")
- newline character
- space character
- tab character

To negate the special meaning of a metacharacter, use one of the quoting mechanisms in the following list.

Item	Description
Backslash	A backslash (\) that is not quoted preserves the literal value of the following character, with the exception of a newline character. If a newline character follows the backslash, then the shell interprets this as line continuation.
Single Quotation Marks	Enclosing characters in single quotation marks (' ') preserves the literal value of each character within the single quotation marks. A single quotation mark cannot occur within single quotation marks. A backslash cannot be used to escape a single quotation mark in a string that is set in single quotation marks. An embedded quotation mark can be created by writing, for example: 'a'\''b', which yields a'b.
Double Quotation Marks	Enclosing characters in double quotation marks (" ") preserves the literal value of all characters within the double quotation marks, with the exception of the dollar sign, backquote, and backslash characters, as follows: **$** The dollar sign retains its special meaning introducing parameter expansion, a form of command substitution, and arithmetic expansion. The input characters within the quoted string that are also enclosed between $(and the matching) will not be affected by the double quotation marks, but define that command whose output replaces the $(...) when the word is expanded. Within the string of characters from an enclosed ${ to the matching }, there must be an even number of unescaped double quotation marks or single quotation marks, if any. A preceding backslash character must be used to escape a literal { or }. **`** The backquote retains its special meaning introducing the other form of command substitution. The portion of the quoted string, from the initial backquote and the characters up to the next backquote that is not preceded by a backslash, defines that command whose output replaces `...` when the word is expanded.  **\** The backslash retains its special meaning as an escape character only when followed by one of the following characters: $, `, ", \, or a newline character.

A double quotation mark must be preceded by a backslash to be included within double quotation marks. When you use double quotation marks, if a backslash is immediately followed by a character that would

be interpreted as having a special meaning, the backslash is deleted, and the subsequent character is taken literally. If a backslash does not precede a character that would have a special meaning, it is left in place unchanged, and the character immediately following it is also left unchanged. For example:

```
"\$"    ->      $
"\a"    ->      \a
```

The following conditions apply to metacharacters and quoting characters in the Korn or POSIX shell:

- The meanings of dollar sign, asterisk ($*) and dollar sign, at symbol ($@) are identical when not quoted, when used as a parameter assignment value, or when used as a file name.
- When used as a command argument, double quotation marks, dollar sign, asterisk, double quotation marks ("$*") is equivalent to "$1*d*$2*d*...", where *d* is the first character of the IFS parameter.
- Double quotation marks, at symbol, asterisk, double quotation marks ("$@") are equivalent to "$1" "$2" ....
- Inside backquotes (``), the backslash quotes the characters backslash (\), single quotation mark ('), and dollar sign ($). If the backquotes occur within double quotation marks (" "), the backslash also quotes the double quotation marks character.
- Parameter and command substitution occurs inside double quotation marks (" ").
- The special meaning of reserved words or aliases is removed by quoting any character of the reserved word. You cannot quote function names or built-in command names.

Restricted Korn shell

The Restricted Korn Shell is used to set up login names and execution environments whose capabilities are more controlled than those of the regular Korn shell.

The **rksh** or **ksh -r** command opens the Restricted Korn Shell. The behavior of these commands is identical to those of the **ksh** command, except that the following actions are not allowed:

- Change the current working directory
- Set the value of the *SHELL*, *ENV*, or *PATH* variables
- Specify the pathname of a command containing a / (slash)
- Redirect output of a command with > (right caret), >| (right caret, pipe symbol), <> (left caret, right caret), or >> (two right carets).

If the Restricted Korn Shell determines that a command to be run is a shell procedure, it uses the Korn shell to run the command. In this way, it is possible to provide an end user with shell procedures that access the full power of the Korn shell while imposing a limited menu of commands. This situation assumes that the user does not have write and execute permissions in the same directory.

If the **File** [*Parameter*] parameter is specified when the Korn shell is started, the shell runs the script file identified by the **File** parameter, including any parameters specified. The script file specified must have read permission. Any `setuid` and `setgid` settings for script files are ignored. The shell then reads the commands. If either the **-c** or **-s** flag is used, do not specify a script file.

When started with the **rksh** command, the shell enforces restrictions after interpreting the `.profile` and `/etc/environment` files. Therefore, the writer of the `.profile` file has complete control over user actions by performing setup actions and leaving the user in an appropriate directory (probably not the login directory). An administrator can create a directory of commands in the `/usr/rbin` directory that the **rksh** command can use by changing the *PATH* variable to contain the directory. If it is started with the **ksh -r** command, the shell applies restrictions when interpreting the `.profile` files.

When called with the **rksh** command, the Restricted Korn Shell reads the user's `.profile` file (`$HOME/.profile`). It acts as the regular Korn shell while doing this, except that an interrupt causes an immediate exit instead of a return to command level.

Reserved words in the Korn shell or POSIX shell

The following reserved words have special meaning to the Korn shell or POSIX shell.

```
!         case    do
done      elif    else
esac      fi      for
function  if      in
select    then    time
until     while   {
}         [[      ]]
```

The reserved words are recognized only when they appear without quotation marks and when the word is used as the following:

- First word of a command
- First word following one of the reserved words other than **case**, **for**, or **in**
- Third word in a **case** or **for** command (only **in** is valid in this case)

Enhanced Korn shell (ksh93)

In addition to the default system Korn shell (`/usr/bin/ksh`), AIX provides an enhanced version available as Korn shell `/usr/bin/ksh93`. This enhanced version is mostly upwardly compatible with the current default version, and includes a few additional features that are not available in Korn shell `/usr/bin/ksh`.

Some scripts might perform differently under Korn shell `ksh93` than under the default shell because variable handling is somewhat different under the two shells.

Note: There is also a restricted version of the enhanced Korn shell available, called `rksh93`.

The following features are not available in Korn shell `/usr/bin/ksh`, but are available in Korn shell `/usr/bin/ksh93`:

Item	Description
Arithmetic enhancements	You can use **libm** functions (math functions typically found in the C programming language), within arithmetic expressions, such as `$ value=$((sqrt(9)))`. More arithmetic operators are available, including the unary **+**, **++**, **--**, and the **?:** construct (for example, "`x ? y : z`"), as well as the **,** (comma) operator. Arithmetic bases are supported up to base 64. Floating point arithmetic is also supported. "**typeset -E**" (exponential) can be used to specify the number of significant digits and "**typeset -F**" (float) can be used to specify the number of decimal places for an arithmetic variable. The **SECONDS** variable now displays to the nearest hundredth of a second, rather than to the nearest second.

Compound variables	Compound variables are supported. A compound variable allows a user to specify multiple values within a single variable name. The values are each assigned with a subscript variable, separated from the parent variable with a period (`.`). For example:
	`$ myvar=( x=1 y=2 )`
	`$ print "${myvar.x}"`
	`1`

Compound assignments	Compound assignments are supported when initializing arrays, both for indexed arrays and associative arrays. The assignment values are placed in parentheses, as shown in the following example:
	`$ numbers=( zero one two three )`
	`$ print ${numbers[0]} ${numbers[3]}`
	`zero three`

Item	Description

Associative arrays	An associative array is an array with a string as an index.
	The **typeset** command used with the **-A** flag allows you to specify associative arrays within ksh93. For example:
	`$ typeset -A teammates`
	`$ teammates=( [john]=smith [mary]=jones )`
	`$ print ${teammates[mary]}`
	`jones`

Variable name references	The **typeset** command used with the **-n** flag allows you to assign one variable name as a reference to another. In this way, modifying the value of a variable will in turn modify the value of the variable that is referenced. For example:
	`$ greeting="hello"`
	`$ typeset -n welcome=greeting  # establishes the reference`
	`$ welcome="hi there"  # overrides previous value`
	`$ print $greeting`
	`hi there`

Parameter expansions	The following parameter-expansion constructs are available: • ${!varname} is the name of the variable itself. • ${!*varname*[@]} names the indexes for the *varname* array. • ${*param*:*offset*} is a substring of *param*, starting at *offset*. • ${param:offset:num} is a substring of *param*, starting at *offset*, for *num* number of characters. • ${@:*offset*} indicates all positional parameters starting at *offset*. • ${@:*offset*:*num*} indicates *num* positional parameters starting at *offset*. • ${*param*/*pattern*/*repl*} evaluates to *param*, with the first occurrence of *pattern* replaced by *repl*. • ${*param*//*pattern*/*repl*} evaluates to *param*, with every occurrence of *pattern* replaced by *repl*. • ${*param*/#*pattern*/*repl*} if *param* begins with *pattern*, then *param* is replaced by *repl*. • ${*param*/%*pattern*/*repl*} if *param* ends with *pattern*, then *param* is replaced by *repl*.

Item	Description

Discipline functions	A discipline function is a function that is associated with a specific variable. This allows you to define and call a function every time that variable is referenced, set, or unset. These functions take the form of *varname.function*, where *varname* is the name of the variable and *function* is the discipline function. The predefined discipline functions are **get**, **set**, and **unset**.
	• The ***varname*.get** function is invoked every time *varname* is referenced. If the special variable **.sh.value** is set within this function, then the value of *varname* is changed to this value. A simple example is the time of day:
	`$ function time.get`
	`> {`
	`>  .sh.value=$(date +%r)`
	`> }`
	`$ print $time`
	`09:15:58 AM`
	`$ print $time  # it will change in a few seconds`
	`09:16:04 AM`
	• The ***varname*.set** function is invoked every time *varname* is set. The **.sh.value** variable is given the value that was assigned. The value assigned to *varname* is the value of **.sh.value** when the function completes. For example:
	`$ function adder.set`
	`> {`
	`>  let .sh.value="`
	`$ {.sh.value} + 1"`
	`> }`
	`$ adder=0`
	`$ echo $adder`
	`1`
	`$ adder=$adder`
	`$ echo $adder`
	`2`
	• The ***varname*.unset** function is executed every time *varname* is unset. The variable is not actually unset unless it is unset within the function itself; otherwise it retains its value.
	Within all discipline functions, the special variable **.sh.name** is set to the name of the variable, while **.sh.subscript** is set to the value of the variables subscript, if applicable.

Function environments	Functions declared with the *function myfunc* format are run in a separate function environment and support local variables. Functions declared as *myfunc* `()` run with the same environment as the parent shell.
Variables	Variables beginning with `.sh.` are reserved by the shell and have special meaning. See the description of Discipline Functions in this table for an explanation of **.sh.name**, **.sh.value**, and **.sh.subscript**. Also available is **.sh.version**, which represents the version of the shell.
Command return values	Return values of commands are as follows: • If the command to be executed is not found, the return value is set to 127. • If the command to be executed is found, but not executable, the return value is 126. • If the command is executed, but is terminated by a signal, the return value is 256 plus the signal number.
PATH search rules	Special built-in commands are searched for first, followed by all functions (including those in **FPATH** directories), followed by other built-ins.

Item	Description
Shell history	The **hist** command allows you to display and edit the shells command history. In the ksh shell, the **fc** command was used. The **fc** command is an alias to **hist**. Variables are *HISTCMD*, which increments once for each command executed in the shells current history, and *HISTEDIT*, which specifies which editor to use when using the **hist** command.
Built-in commands	The enhanced Korn shell contains the following built-in commands: • The **builtin** command lists all available built-in commands. • The **printf** command works in a similar manner as the **printf()** C library routine. See the **printf** command. • The **disown** blocks the shell from sending a SIGHUP to the specified command. • The **getconf** command works in the same way as the stand-alone command **/usr/bin/getconf**. See the **getconf** command. • The **read** built-in command has the following flags: – **read -d {*char*}** allows you to specify a character delimiter instead of the default newline. – **read -t {*seconds*}** allows you to specify a time limit, in seconds, after which the **read** command will time out. If **read** times out, it will return FALSE. • The **exec** built-in command has the following flags: – **exec -a {*name*} {*cmd*}** specifies that argument 0 of *cmd* be replaced with *name*. – **exec -c {*cmd*}** tells **exec** to clear the environment before executing *cmd*. • The **kill** built-in command has the following flags: – **kill -n {*signum*}** is used for specifying a signal number to send to a process, while **kill -s {*signame*}** is used to specify a signal name. – **kill -l**, with no arguments, lists all signal names but not their numbers. • The **whence** built-in command has the following flags: – The **-a** flag displays all matches, not only the first one found. – The **-f** flag tells **whence** not to search for any functions. • An escape character sequence is used for use by the **print** and **echo** commands. The Esc (Escape) key can be represented by the sequence \E. • All regular built-in commands recognize the **-?** flag, which shows the syntax for the specified command. • The **getopts** built-in requires **optstring** to contain a leading **+** to allow options beginning with a **+** symbol.

Item	Description
Audit support	To enable the auditing feature in the enhanced Korn shell, complete the following steps: • To enable the auditing feature in the **ksh93u** command, the */etc/ksh_audit* file must have an output filename an identification value for the user. For example, `/tmp/ksh_auditfile;205;0`. • You can configure the auditing feature to generate a detailed record for every executable command. You can use the record to monitor, track, and audit activities for one or more users on a system, including the system administrators. • To enable the auditing feature in the **ksh93** command, the file */etc/ksh_audit* must be created and configured by specifying the location of the *audit_log* file and an identification value for the user. You can separate the identification values by using a semicolon. For example, the text */tmp/ksh_auditfile;0;100;205* in the */etc/ksh_audit* file refers to the location of the log file (`/tmp/ksh_auditfile`), and the user identification values which are, *0 (root)*, *100*, and *205* respectively. **Note:** • The users awaiting audit must have *Read* and *Write* permissions for the configuration and log files. • When the `EXTENDED_HISTORY` environment variable is switched on, the audit entries will have a timestamp information with non-printable characters at the end of the entry. By default, the `EXTENDED_HISTORY` environment variable is turned on. However, you can manually turn it on by running the **`export EXTENDED_HISTORY=ON`** command. To disable the `EXTENDED_HISTORY` environment variable, you can run the **`unset EXTENDED_HISTORY`** command. • The `EXTENDED_HISTORY` environment variable allows you to view a detailed tracking report for all commands that are executed in the shell.

Item	Description
Other miscellaneous differences between Korn shell ksh and Korn shell ksh93	Other differences are: • With Korn shell ksh93, you cannot export functions using the **typeset -fx** built-in command. • With Korn shell ksh93, you cannot export an alias using the **alias -x** built-in command. • With Korn shell ksh93, a dollar sign followed by a single quote ($') is interpreted as an ANSI C string. You must quote the dollar sign (\"$\"') to get the old (ksh) behavior. • Argument parsing logic for Korn shell ksh93 built-in commands has been changed. The undocumented combinations of argument parsing to Korn shell ksh built-in commands do not work in Korn shell ksh93. For example, **typeset -4i** works similar to **typeset -i4** in Korn shell ksh, but does not work in Korn shell ksh93. • With Korn shell ksh93, command substitution and arithmetic expansion is performed on special environment variables PS1, PS3, and ENV while expanding. Therefore, you must escape the grave symbol (`) and the dollar sign and opening parenthesis symbols ($() using a backslash (\) to retain the old behavior. For example, Korn shell ksh literally assigns x=$'*name*\t*operator*' as $*name*\t*operator*; Korn shell ksh93 expands \t and assigns it as *name*<\t expanded>*operator*. To preserve the Korn shell ksh behavior, you must quote $. For example, x="$"'*name*\t*operator*'. • The *ERRNO* variable has been removed in Korn shell ksh93. • In Korn shell ksh93, file names are not expanded for non-interactive shells after the redirection symbol. • With Korn shell ksh93, you must use the **-t** option of the **alias** command to display tracked aliases. The tracked alias feature is now obsolete, so the displayed aliases might not be tracked. • With Korn shell ksh93, in emacs mode, Ctrl+T swaps the current and previous character. With ksh, Ctrl+T swaps the current and next character. • Korn shell ksh93 does not allow unbalanced parentheses within ${*name operator value*}. For example, ${*name*-(} needs an escape such as ${*name*-\(} to work in both versions. • With Korn shell ksh93, the **kill -l** command lists only the signal names, not their numerical values.

Exit status in the Korn shell or POSIX shell

Errors detected by the shell, such as syntax errors, cause the shell to return a nonzero exit status. Otherwise, the shell returns the exit status of the last command carried out.

The shell reports detected runtime errors by printing the command or function name and the error condition. If the number of the line on which an error occurred is greater than 1, then the line number is also printed in **[ ]** (brackets) after the command or function name.

For a noninteractive shell, an error encountered by a special built-in or other type of command will cause the shell to write a diagnostic message as shown in the following table:

Error	Special Built-In	Other Utilities
Shell language syntax error	will exit	will exit
Utility syntax error (option or operand error)	will exit	will not exit
Redirection error	will exit	will not exit
Variable assignment error	will exit	will not exit

Error	Special Built-In	Other Utilities
Expansion error	will exit	will exit
Command not found	not applicable	may exit
Dot script not found	will exit	not applicable

If any of the errors shown as "will (may) exit" occur in a subshell, the subshell will (may) exit with a nonzero status, but the script containing the subshell will not exit because of the error.

In all cases shown in the table, an interactive shell will write a diagnostic message to standard error, without exiting.

Parameters in the Korn shell

Korn shell parameters are discussed below.

A parameter is defined as the following:

- Identifier of any of the characters asterisk (*), at sign (@), pound sign (#), question mark (?), hyphen (-), dollar sign ($), and exclamation point (!). These are called *special parameters*.
- Argument denoted by a number (*positional parameter*)
- Parameter denoted by an identifier, with a value and zero or more attributes (*named parameter/ variables*).

The **typeset** special built-in command assigns values and attributes to named parameters. The attributes supported by the Korn shell are described with the **typeset** special built-in command. Exported parameters pass values and attributes to the environment.

The value of a named parameter is assigned by:

```
Name=Value [ Name=Value ] ...
```

If the **-i** integer attribute is set for the **Name** parameter, then the **Value** parameter is subject to arithmetic evaluation.

The shell supports a one-dimensional array facility. An element of an array parameter is referenced by a subscript. A subscript is denoted by an arithmetic expression enclosed by brackets `[ ]`. To assign values to an array, use `set -A` *Name Value* ... . The value of all subscripts must be in the range of 0 through 511. Arrays need not be declared. Any reference to a named parameter with a valid subscript is legal and an array will be created, if necessary. Referencing an array without a subscript is equivalent to referencing the element 0.

Positional parameters are assigned values with the **set** special command. The **$0** parameter is set from argument `0` when the shell is invoked. The $ character is used to introduce parameters that can be substituted.

Related concepts

Shell startup
You can start the Korn shell with the **ksh** command, **psh** command (POSIX shell), or the **exec** command.

Korn shell functions
The **function** reserved word defines shell functions. The shell reads and stores functions internally. Alias names are resolved when the function is read. The shell executes functions in the same manner as commands, with the arguments passed as positional parameters.

Arithmetic evaluation in the Korn shell or POSIX shell
The Korn shell or POSIX shell regular built-in **let** command enables you to perform integer arithmetic.

Related reference

Korn shell compound commands

A compound command can be a list of simple commands or a pipeline, or it can begin with a reserved word. Most of the time, you will use compound commands such as **if**, **while**, and **for** when you are writing shell scripts.

Parameter substitution in the Korn shell or POSIX shell
The Korn shell, or POSIX shell, lets you perform parameter substitutions.

The following are substitutable parameters:

Item	Description
${*Parameter*}	The shell reads all the characters from the ${ to the matching } as part of the same word, even if that word contains braces or metacharacters. The value, if any, of the specified parameter is substituted. The braces are required when the *Parameter* parameter is followed by a letter, digit, or underscore that is not to be interpreted as part of its name, or when a named parameter is subscripted. If the specified parameter contains one or more digits, it is a *positional parameter*. A positional parameter of more than one digit must be enclosed in braces. If the value of the variable is * or @), each positional parameter, starting with $1, is substituted (separated by a field separator character). If an array identifier with a subscript * or @ is used, then the value for each of the elements (separated by a field separator character) is substituted.
${#*Parameter*}	If the value of the *Parameter* parameter is * or @, the number of positional parameters is substituted. Otherwise, the length specified by the *Parameter* parameter is substituted.
${#*Identifier*[*]}	The number of elements in the array specified by the *Identifier* parameter is substituted.
${*Parameter*:-*Word*}	If the *Parameter* parameter is set and is not null, then its value is substituted; otherwise, the value of the *Word* parameter is substituted.
${*Parameter*:=*Word*}	If the *Parameter* parameter is not set or is null, then it is set to the value of the *Word* parameter. Positional parameters cannot be assigned in this way.
${*Parameter*:?*Word*}	If the *Parameter* parameter is set and is not null, then substitute its value. Otherwise, print the value of the *Word* variable and exit from the shell. If the *Word* variable is omitted, then a standard message is printed.
${*Parameter*:+*Word*}	If the *Parameter* parameter is set and is not null, then substitute the value of the *Word* variable.
${*Parameter*#*Pattern*} | ${*Parameter*##*Pattern*}	If the specified shell *Pattern* parameter matches the beginning of the value of the *Parameter* parameter, then the value of this substitution is the value of the *Parameter* parameter with the matched portion deleted. Otherwise, the value of the *Parameter* parameter is substituted. In the first form, the smallest matching pattern is deleted. In the second form, the largest matching pattern is deleted.

Item	Description
${*Parameter%Pattern*} | ${*Parameter%%Pattern*}	If the specified shell *Pattern* matches the end of the value of the *Parameter* variable, then the value of this substitution is the value of the *Parameter* variable with the matched part deleted. Otherwise, substitute the value of the *Parameter* variable. In the first form, the smallest matching pattern is deleted; in the second form, the largest matching pattern is deleted. In the previous expressions, the *Word* variable is not evaluated unless it is to be used as the substituted string. Thus, in the following example, the **pwd** command is executed only if the **-d** flag is not set or is null: `echo ${d:-$(pwd)}`

Note: If the : is omitted from the previous expressions, the shell checks only whether the *Parameter* parameter is set.

Related concepts

Unattended terminals

All systems are vulnerable if terminals are left logged in and unattended. The most serious problem occurs when a system manager leaves a terminal unattended that has been enabled with root authority. In general, users should log out anytime they leave their terminals.

Predefined special parameters in the Korn shell or POSIX shell

Some parameters are set automatically by the Korn shell or POSIX shell.

The following parameters are automatically set by the shell:

Item	Description
@	Expands the positional parameters, beginning with $1. Each parameter is separated by a space. If you place " around $@, the shell considers each positional parameter a separate string. If no positional parameters exist, the shell expands the statement to an unquoted null string.
*****	Expands the positional parameters, beginning with $1. The shell separates each parameter with the first character of the **IFS** parameter value. If you place " around $*, the shell includes the positional parameter values in double quotation marks. Each value is separated by the first character of the **IFS** parameter.
#	Specifies the number (in decimals) of positional parameters passed to the shell, not counting the name of the shell procedure itself. The **$#** parameter thus yields the number of the highest-numbered positional parameter that is set. One of the primary uses of this parameter is to check for the presence of the required number of arguments.
-	Supplies flags to the shell on invocation or with the **set** command.
?	Specifies the exit value of the last command executed. Its value is a decimal string. Most commands return 0 to indicate successful completion. The shell itself returns the current value of the **$?** parameter as its exit value.

Item	Description

$	Identifies the process number of this shell. Because process numbers are unique among all existing processes, this string of up to 5 digits is often used to generate unique names for temporary files.
	
	The following example illustrates the recommended practice of creating temporary files in a directory used only for that purpose:
	
	`temp=$HOME/temp/$$`
	`ls >$temp`
	`.`
	`.`
	`.`
	`rm $temp`

!	Specifies the process number of the most recent background command invoked.
zero (0)	Expands to the name of the shell or shell script.

File name substitution in the Korn shell or POSIX shell

The Korn shell, or POSIX shell, performs file name substitution by scanning each command word specified by the *Word* variable for certain characters.

If a command word includes the ***)**, **?** or **[** characters, and the **-f** flag has not been set, the shell regards the word as a pattern. The shell replaces the word with file names, sorted according to the collating sequence in effect in the current locale, that match that pattern. If the shell does not find a file name to match the pattern, it does not change the word.

When the shell uses a pattern for file name substitution, the **.** and **/** characters must be matched explicitly.

Note: The Korn shell does not treat these characters specially in other instances of pattern matching.

These pattern-matching characters indicate the following substitutions:

Item	Description
*****	Matches any string, including the null string.
?	Matches any single character.
[...**]**	Matches any one of the enclosed characters. A pair of characters separated by a hyphen (-) matches any character lexically within the inclusive range of that pair, according to the collating sequence in effect in the current locale. If the first character following the opening `[` is `!`, then any character not enclosed is matched. A hyphen (-) can be included in the character set by putting it as the first or last character.

You can also use the `[:charclass:]` notation to match file names within a range indication. This format instructs the system to match any single character belonging to `class`. The definition of which characters constitute a specific character class is present through the **LC_CTYPE** category of the setlocale subroutine. All character classes specified in the current locale are recognized.

The names of some of the character classes are as follows:

- **alnum**
- **alpha**
- **cntrl**
- **digit**
- **graph**
- **lower**
- **print**
- **punct**

- **space**
- **upper**
- **xdigit**

For example, `[[:upper:]]` matches any uppercase letter.

The Korn shell supports file name expansion based on collating elements, symbols, or equivalence classes.

A *PatternList* is a list of one or more patterns separated from each other with a `|`. Composite patterns are formed with one or more of the following:

Item	Description
?(*PatternList***)**	Optionally matches any one of the given patterns
(PatternList***)**	Matches zero or more occurrences of the given patterns
+(*PatternList***)**	Matches one or more occurrences of the given patterns
@(*PatternList***)**	Matches exactly one of the given patterns
!(*PatternList***)**	Matches anything, except one of the given patterns

Pattern matching has some restrictions. If the first character of a file name is a dot (`.`), it can be matched only by a pattern that also begins with a dot. For example, `*` matches the file names `myfile` and `yourfile` but not the file names `.myfile` and `.yourfile`. To match these files, use a pattern such as the following:

```
.*file
```

If a pattern does not match any file names, then the pattern itself is returned as the result of the attempted match.

File and directory names should not contain the characters `*`, `?`, `[`, or `]` because they can cause infinite recursion (that is, infinite loops) during pattern-matching attempts.

Quote removal
Some characters will be removed if they are not quoted.

The quote characters, backslash (`\`), single quote (`'`), and double quote (`"`) that were present in the original word will be removed unless they have themselves been quoted.

Input and output redirection in the Korn shell or POSIX shell

Before the Korn shell executes a command, it scans the command line for redirection characters. These special notations direct the shell to redirect input and output.

Redirection characters can appear anywhere in a simple command or can precede or follow a command. They are not passed on to the invoked command.

The shell performs command and parameter substitution before using the **Word** or **Digit** parameter except as noted. File name substitution occurs only if the pattern matches a single file and blank interpretation is not performed.

Item	Description
<*Word*	Uses the file specified by the **Word** parameter as standard input (file descriptor 0).

Item	Description
>*Word*	Uses the file specified by the **Word** parameter as standard output (file descriptor 1). If the file does not exist, the shell creates it. If the file exists and the **noclobber** option is on, an error results; otherwise, the file is truncated to zero length. **Note:** When multiple shells have the **noclobber** option set and they redirect output to the same file, there could be a race condition, which might result in more than one of these shell processes writing to the file. The shell does not detect or prevent such race conditions.
>|*Word*	Same as the **>***Word* command, except that this redirection statement overrides the **noclobber** option.
> >*Word*	Uses the file specified by the **Word** parameter as standard output. If the file currently exists, the shell appends the output to it (by first seeking the end-of-file character). If the file does not exist, the shell creates it.
<>*Word*	Opens the file specified by the **Word** parameter for reading and writing as standard input.
<<[**-**]*Word*	Reads each line of shell input until it locates a line containing only the value of the **Word** parameter or an end-of-file character. The shell does not perform parameter substitution, command substitution, or file name substitution on the file specified. The resulting document, called a here document, becomes the standard input. If any character of the **Word** parameter is quoted, no interpretation is placed upon the characters of the document.

The *here* document is treated as a single word that begins after the next newline character and continues until there is a line containing only the delimiter, with no trailing blank characters. Then the next here document, if any, starts. The format is as follows:

```
[n]<<word
   here document
delimiter
```

If any character in *word* is quoted, the delimiter is formed by removing the quote on *word*. The *here* document lines will not be expanded. Otherwise, the delimiter is the *word* itself. If no characters in *word* are quoted, all lines of the here document will be expanded for parameter expansion, command substitution, and arithmetic expansion.

The shell performs parameter substitution for the redirected data. To prevent the shell from interpreting the \, $, and single quotation mark (') characters and the first character of the **Word** parameter, precede the characters with a \ character.

If a hyphen (-) is appended to <<, the shell strips all leading tabs from the **Word** parameter and the document.

Item	Description
<&*Digit*	Duplicates standard input from the file descriptor specified by the **Digit** parameter
>& *Digit*	Duplicates standard output in the file descriptor specified by the **Digit** parameter
<&-	Closes standard input
>&-	Closes standard output
<&p	Moves input from the co-process to standard input
>&p	Moves output to the co-process to standard output

If one of these redirection options is preceded by a digit, then the file descriptor number referred to is specified by the digit (instead of the default 0 or 1). In the following example, the shell opens file descriptor 2 for writing as a duplicate of file descriptor 1:

```
... 2>&1
```

The order in which redirections are specified is significant. The shell evaluates each redirection in terms of the (*FileDescriptor*, *File*) association at the time of evaluation. For example, in the statement:

```
... 1>File 2>&1
```

the file descriptor 1 is associated with the file specified by the **File** parameter. The shell associates file descriptor 2 with the file associated with file descriptor 1 (*File*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had previously been) and file descriptor 1 would be associated with the file specified by the **File** parameter.

If a command is followed by an ampersand (&) and job control is not active, the default standard input for the command is the empty file `/dev/null`. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input and output specifications.

Related concepts

Input and output redirection
The AIX operating system allows you to manipulate the input and output (I/O) of data to and from your system by using specific I/O commands and symbols.

Related tasks

Redirecting output to inline input (here) documents
You can redirect output to inline input (here) documents.

Coprocess facility

The Korn shell, or POSIX shell, allows you to run one or more commands as background processes. These commands, run from within a shell script, are called *coprocesses*.

Designate a coprocess by placing the |& operator after a command. Both standard input and output of the command are piped to your script.

A coprocess must meet the following restrictions:

- Include a newline character at the end of each message
- Send each output message to standard output
- Clear its standard output after each message

The following example demonstrates how input is passed to and returned from a coprocess:

```
echo "Initial process"
./FileB.sh |&
read -p a b c d
echo "Read from coprocess: $a $b $c $d"
print -p "Passed to the coprocess"
read -p a b c d
echo "Passed back from coprocess: $a $b $c $d"
```

```
FileB.sh
   echo "The coprocess is running"
   read a b c d
   echo $a $b $c $d
```

The resulting standard output is as follows:

```
Initial process
Read from coprocess: The coprocess is running
Passed back from coprocess: Passed to the coprocess
```

Use the **print -p** command to write to the coprocess. Use the **read -p** command to read from the coprocess.

Related concepts

Korn shell or POSIX shell commands
The Korn shell is an interactive command interpreter and command programming language. It conforms to the Portable Operating System Interface for Computer Environments (POSIX), an international standard for operating systems.

Redirection of coprocess input and output
The standard input and output of a coprocess is reassigned to a numbered file descriptor by using I/O redirection.

For example, the command:

```
exec 5>&p
```

moves the input of the coprocess to file descriptor 5.

After this coprocess has completed, you can use standard redirection syntax to redirect command output to the coprocess. You can also start another coprocess. Output from both coprocesses is connected to the same pipe and is read with the **read -p** command. To stop the coprocess, type the following:

```
read -u5
```

Korn shell or POSIX shell built-in commands

Special commands are built in to the Korn shell and POSIX shell and executed in the shell process.

Unless otherwise indicated, the output is written to file descriptor 1 and the exit status is zero (0) if the command does not contain any syntax errors. Input and output redirection is permitted. There are two types of built-in commands: *special built-in commands* and *regular built-in commands*.

Special built-in commands differ from regular built-in commands in the following ways:

- A syntax error in a special built-in command might cause the shell executing the command to end. This does not happen if you have a syntax error in a regular built-in command. If a syntax error in a special built-in command does not end the shell program, the exit value is nonzero.
- Variable assignments specified with special built-in commands remain in effect after the command completes.
- I/O redirections are processed after parameter assignments.

In addition, words that are in the form of a parameter assignment following the **export**, **readonly**, and **typeset** special commands are expanded with the same rules as a parameter assignment. Tilde substitution is performed after the =, and word-splitting and file name substitution are not performed.

Related concepts

Korn shell or POSIX shell commands
The Korn shell is an interactive command interpreter and command programming language. It conforms to the Portable Operating System Interface for Computer Environments (POSIX), an international standard for operating systems.

Korn shell functions
The **function** reserved word defines shell functions. The shell reads and stores functions internally. Alias names are resolved when the function is read. The shell executes functions in the same manner as commands, with the arguments passed as positional parameters.

Related reference

List of Korn shell or POSIX shell special built-in commands
Special commands are built into the Korn shell and POSIX shell and executed in the shell process.

Korn shell or POSIX shell regular built-in commands

The following is a list of the Korn shell or POSIX shell regular built-in commands.

Special built-in command descriptions for the Korn shell or POSIX shell
Special commands are built into the Korn shell and POSIX shell and executed in the shell process.

The special built-in commands of the Korn shell are described below:

:	eval	newgrp	shift
.	exec	readonly	times
break	exit	return	trap
continue	export	set	typeset
			unset

Item	Description
: [*Argument* ...]	Expands only arguments. It is used when a command is necessary, as in the *then* condition of an **if** command, but nothing is to be done by the command.
. *File* [*Argument* ...]	Reads the complete specified file and then executes the commands. The commands are executed in the current shell environment. The search path specified by the *PATH* variable is used to find the directory containing the specified file. If any arguments are specified, they become the positional parameters. Otherwise, the positional parameters are unchanged. The exit status is the exit status of the most recent command executed. See "Parameter substitution in the Korn shell or POSIX shell" on page 225 for more information on positional parameters. **Note:** The *.File* [*Argument* ...] command reads the entire file before any commands are carried out. Therefore, the **alias** and **unalias** commands in the file do not apply to any functions defined in the file.
break [*n*]	Exits from the enclosing **for**, **while**, **until**, or **select** loop, if one exists. If you specify the *n* parameter, the command breaks the number of levels specified by the *n* parameter. The value of *n* is any integer equal to or greater than 1.
continue [*n*]	Resumes the next iteration of the enclosing **for**, **while**, **until**, or **select** loop. If you specify the *n* parameter, the command resumes at the n^{th} enclosing loop. The value of *n* is any integer equal to or greater than 1.
eval [*Argument* ...]	Reads the specified arguments as input to the shell and executes the resulting command or commands.
exec [*Argument* ...]	Executes the command specified by the argument in place of this shell (without creating a new process). Input and output arguments can appear and affect the current process. If you do not specify an argument, the **exec** command modifies file descriptors as prescribed by the input and output redirection list. In this case, any file descriptor numbers greater than 2 that are opened with this mechanism are closed when invoking another program.
exit [*n*]	Exits the shell with the exit status specified by the *n* parameter. The *n* parameter must be an unsigned decimal integer with range 0-255. If you omit the *n* parameter, the exit status is that of the most recent command executed. An end-of-file character also exits the shell unless the **ignoreeof** option of the **set** special command is turned on.

Item	Description
export -p [*Name*[**=** *Value*]] ...	Marks the specified names for automatic export to the environment of subsequently executed commands. **-p** writes to standard output the names and values of all exported variables, in the following format: `"export %s= %s\n", <name> <value>`
newgrp [*Group*]	Equivalent to the **exec/usr/bin/newgrp** [*Group*] command. **Note:** This command does not return.
readonly -p [*Name*[**=** *Value*]] ...	Marks the names specified by the *Name* parameter as read-only. These names cannot be changed by subsequent assignment. **-p** writes to standard output the names and values of all exported variables, in the following format: `"export %s= %s\n", <name> <value>`
return [*n*]	Causes a shell function to return to the invoking script. The return status is specified by the *n* parameter. If you omit the *n* parameter, the return status is that of the most recent command executed. If you invoke the **return** command outside of a function or a script, then it is the same as an **exit** command.

Item	Description
set [**+** |**-abCefhkmnostuvx**] [**+** |**-o** *Option*]... [**+** |**-A***Name*] [*Argument* ...]	If no options or arguments are specified, the **set** command writes the names and values of all shell variables in the collation sequence of the current locale. When options are specified, they will set or unset attributes of the shell, described as follows: **-A** Array assignment. Unsets the *Name* parameter and assigns values sequentially from the specified *Argument* parameter list. If the **+A** flag is used, the *Name* parameter is not unset first. **-a** Automatically exports all subsequent parameters that are defined. **-b** Notifies the user asynchronously of background job completions. **-C** Equivalent to `set -o noclobber`. **-e** Executes the ERR trap, if set, and exits if a command has a nonzero exit status unless the simple command is: + contained in an && or || list + the command immediately following if, while or until + contained in the pipeline following ! This mode is disabled while reading profiles. **-f** Disables file name substitution. **-h** Designates each command as a tracked alias when first encountered. **-k** Places all parameter-assignment arguments in the environment for a command, not only those arguments that precede the command name. **-m** Runs background jobs in a separate process and prints a line upon completion. The exit status of background jobs is reported in a completion message. On systems with job control, this flag is turned on automatically for interactive shells. For more information, see "Job control in the Korn shell or POSIX shell" on page 243. **-n** Reads commands and checks them for syntax errors, but does not execute them. This flag is ignored for interactive shells.

Item	Description
	-o *Option* Prints current option settings and an error message if you do not specify an argument. You can set more than one option on a single **ksh** command line. If the **+o** flag is used, the specified option is unset. When arguments are specified, they will cause positional parameters to be set or unset. Arguments, as specified by the *Option* variable, can be one of the following: **allexport** Same as the **-a** flag. **bgnice** Runs all background jobs at a lower priority. This is the default mode. **emacs** Enters an emacs-style inline editor for command entry. **errexit** Same as the **-e** flag. **gmacs** Enters a gmacs-style inline editor for command entry. **ignoreeof** Does not exit the shell when it encounters an end-of-file character. To exit the shell, you must use the **exit** command or press the Ctrl-D key sequence more than 11 times. **keyword** Same as the **-k** flag. **Note:** This flag is for backward compatibility with the Bourne shell only. Its use is strongly discouraged. **markdirs** Appends a backslash / to all directory names that are a result of file name substitution. **monitor** Same as the **-m** flag. **noclobber** Prevents redirection from truncating existing files. When you specify this option, a vertical bar must follow the redirection symbol (>|) to truncate a file. **noexec** Same as the **-n** flag. **noglob** Same as the **-f** flag. **nolog** Prevents function definitions in `.profile` and $ENV files from being saved in the history file. **nounset** Same as the **-u** flag. **privileged** Same as the **-p** flag.

Item	Description

trackall
Same as the **-h** flag.

verbose
Same as the **-v** flag.

vi
Enters the insert mode of a vi-style inline editor for command entry. Entering escape character 033 puts the editor into the move mode. A return sends the line.

viraw
Processes each character as it is typed in vi mode.

xtrace
Same as the **-x** flag.

-p
Disables processing of the `$HOME/.profile` file and uses the `/etc/suid_profile` file instead of the ENV file. This mode is enabled whenever the effective user ID (UID) or group ID (GID) is not equal to the real UID or GID. Turning off this option sets the effective UID or GID to the real UID and GID.

Note: The system does not support the **-p** option because the operating system does not support **setuid** shell scripts.

-s
Sorts the positional parameters lexicographically.

-t
Exits after reading and executing one command.

Note: This flag is for backward compatibility with the Bourne shell only. Its use is strongly discouraged.

-u
Treats unset parameters as errors when substituting.

-v
Prints shell input lines as they are read.

-x
Prints commands and their arguments as they are executed.

-
Turns off the **-x** and **-v** flags and stops examining arguments for flags.

—
Prevents any flags from being changed. This option is useful in setting the *$1* parameter to a value beginning with `-`. If no arguments follow this flag, the positional parameters are not set.

Preceding any of the **set** command flags with a + rather than a `-` turns off the flag. You can use these flags when you invoke the shell. When '`set +o`' is invoked without any arguments, it displays the current option settings in a format that is suitable for re-input to the shell as commands that achieve the same option setting. The current set of flags is found in the **$-** parameter. Unless you specify the **-A** flag, the remaining arguments are positional parameters and are assigned, in order, to $1, $2, `...`, and so on. If no arguments are given, the names and values of all named parameters are printed to standard output.

Item	Description
shift [*n*]	Renames the positional parameters, beginning with $n+1 ... through $1 .... The default value of the *n* parameter is 1. The *n* parameter is any arithmetic expression that evaluates to a nonnegative number less than or equal to the $# parameter.
times	Prints the accumulated user and system times for the shell and for processes run from the shell.
trap [*Command*] [*Signal*] ...	Runs the specified command when the shell receives the specified signal or signals. The *Command* parameter is read once when the trap is set and once when the trap is taken. The *Signal* parameter can be given as a number or as the name of the signal. Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. If the command is -, all traps are reset to their original values. If you omit the command and the first signal is a numeric signal number, then the **ksh** command resets the value of the *Signal* parameter or parameters to the original values. **Note:** If you omit the command and the first signal is a symbolic name, the signal is interpreted as a command. If the value of the *Signal* parameter is the **ERR** signal, the specified command is carried out whenever a command has a nonzero exit status. If the signal is **DEBUG**, then the specified command is carried out after each command. If the value of the *Signal* parameter is the **0** or **EXIT** signal and the **trap** command is executed inside the body of a function, the specified command is carried out after the function completes. If the *Signal* parameter is **0** or **EXIT** for a **trap** command set outside any function, the specified command is carried out on exit from the shell. **Note:** If a script receives a SIGINT signal within a function, the EXIT signal cannot be trapped when the shell exits. The **trap** command with no arguments prints a list of commands associated with each signal number. If the command specified is NULL, indicated as "" (empty quotes), then the **ksh** command will ignore the signal. For more information about how the Korn shell or the POSIX shell reads a character as a regular character, see "Quotation of characters in the Korn shell or POSIX shell" on page 215. For a complete list of *Signal* parameter values used in the **trap** command without the **SIG** prefix, see the **sigaction**, **sigvec**, or **signal** subroutine.

Item	Description
typeset [**+HLRZfilrtux** [*n*]] [*Name*[**=***Value*]] ...	Sets attributes and values for shell parameters. When invoked inside a function, a new instance of the *Name* parameter is created. The parameter value and type are restored when the function completes. You can specify the following flags with the **typeset** command: **-H** Provides AIX-to-host-file mapping on non-AIX machines. **-L** Left-justifies and removes leading blanks from the *Value* parameter. If the *n* parameter has a nonzero value, it defines the width of the field; otherwise, it is determined by the width of the value of its first assignment. When the parameter is assigned, it is filled on the right with blanks or truncated, if necessary, to fit into the field. Leading zeros are removed if the **-Z** flag is also set. The **-R** flag is turned off. **-R** Right-justifies and fills with leading blanks. If the *n* parameter has a nonzero value, it defines the width of the field; otherwise, it is determined by the width of the value of its first assignment. The field remains filled with blanks or is truncated from the end if the parameter is reassigned. The **L** flag is turned off. **-Z** Right-justifies and fills with leading zeros if the first nonblank character is a digit and the **-L** flag has not been set. If the *n* parameter has a nonzero value, it defines the width of the field; otherwise, it is determined by the width of the value of its first assignment. **-f** Indicates that the names refer to function names rather than parameter names. No assignments can be made and the only other valid flags are **-t**, **-u**, and **-x** . The **-t** flag turns on execution tracing for this function. The **-u** flag causes this function to be marked undefined. The *FPATH* variable is searched to find the function definition when the function is referenced. The **-x** flag allows the function definition to remain in effect across shell scripts that are not a separate invocation of the **ksh** command. **-i** Identifies the parameter as an integer, making arithmetic faster. If the *n* parameter has a nonzero value, it defines the output arithmetic base; otherwise, the first assignment determines the output base. **-l** Converts all uppercase characters to lowercase. The **-u** uppercase conversion flag is turned off. **-r** Marks the names specified by the *Name* parameter as read-only. These names cannot be changed by subsequent assignment.

Item	Description
	-t Tags the named parameters. Tags can be defined by the user and have no special meaning to the shell. **-u** Converts all lowercase characters to uppercase characters. The **-l** lowercase flag is turned off. **-x** Marks the name specified by the *Name* parameter for automatic export to the environment of subsequently executed commands. Using a + rather than a - turns off the **typeset** command flags. If you do not specify *Name* parameters but do specify flags, a list of names (and optionally the values) of the parameters that have these flags set is printed. (Using a + rather than a - keeps the values from being printed.) If you do not specify any names or flags, the names and attributes of all parameters are printed.
unset [**-fv**] *Name* ...	Unsets the values and attributes of the parameters given by the list of names. If **-v** is specified, *Name* refers to a variable name, and the shell will unset it and remove it from the environment. Read-only variables cannot be unset. Unsetting the *ERRNO*, *LINENO*, *MAILCHECK*, *OPTARG*, *OPTIND*, *RANDOM*, *SECONDS*, *TMOUT*, and underscore (_) variables removes their special meanings even if they are subsequently assigned. If the **-f** flag is set, then *Name* refers to a function name, and the shell will unset the function definition.

Regular built-in command descriptions for the Korn shell or POSIX shell
The built-in commands for the Korn or POSIX shells are described here.

The Korn shell provides the following regular built-in commands:

```
alias           fg              print           ulimit
bg              getopts         pwd             umask
cd              jobs            read            unalias
command         kill            setgroups       wait
echo            let             setsenv            test            whence
fc
```

Item	Description
alias [**-t**] [**-x**] [*AliasName*[**=** *String*]] ...	Creates or redefines alias definitions or writes existing alias definitions to standard output. For more information, see the **alias** command.
bg [*JobID*...]	Puts each specified job into the background. The current job is put in the background if a *JobID* parameter is not specified. See "Job control in the Korn shell or POSIX shell" on page 243 for more information about job control. For more information about running jobs in the background, see the **bg** command.
cd [*Argument*]	

Item	Description
cd *Old New*	This command can be in either of two forms. In the first form, it changes the current directory to the one specified by the *Argument* parameter. If the value of the *Argument* parameter is a hyphen (`-`), the directory is changed to the previous directory. The `HOME` shell variable is the default value of the *Argument* parameter. The `PWD` variable is set to the current directory. The `CDPATH` shell variable defines the search path for the directory containing the value of the *Argument* parameter. Alternative directory names are separated by a colon (`:`). The default path is null, specifying the current directory. The current directory is specified by a null path name, which appears immediately after the equal sign or between the colon delimiters anywhere in the path list. If the specified argument begins with a slash (`/`), the search path is not used. Otherwise, each directory in the path is searched for the argument. The second form of the **cd** command substitutes the string specified by the *New* variable for the string specified by the *Old* variable in the current directory name, `PWD`, and tries to change to this new directory.
command [**-p**] *CommandName* [*Argument* ...]	
command [**-v** | **-V**] *CommandName*	Causes the shell to treat the specified command and arguments as a simple command, suppressing shell-function lookup. For more information, see the **command** command.
echo [*String* ...]	Writes character strings to standard output. See the **echo** command for usage and description. The **-n** flag is not supported.
fc [**-r**] [**-e** *Editor*] [*First* [*Last*]] **fc -l** [**-n**] [**-r**] [*First* [*Last*]]	
fc -s [*Old***=** *New*] [*First*]	Displays the contents of your command history file or invokes an editor to modify and re-execute commands previously entered in the shell. For more information, see the **fc** command.
fg [*JobID*]	Brings each job specified into the foreground. If you do not specify any jobs, the command brings the current job into the foreground. For more information about running jobs in the foreground, see the **fg** command.
getopts *OptionString Name* [*Argument* ...]	Checks the *Argument* parameter for legal options. For more information, see the **getopts** command.
jobs [**-l** | **-n** | **-p**] [*JobID* ...]	Displays the status of jobs started in the current shell environment. If no specific job is specified with the *JobID* parameter, status information for all active jobs is displayed. If a job termination is reported, the shell removes that job's process ID from the list of those known by the current shell environment. For more information, see the **jobs** command.

Item	Description
kill [**-s** { *SignalName* | *SignalNumber* }] *ProcessID...* **kill** [-*SignalName* | -*SignalNumber*] *ProcessID...*	Sends a signal (by default, the SIGTERM signal) to a running process. This default action normally stops processes. If you want to stop a process, specify the process ID (PID) in the *ProcessID* variable. The shell reports the PID of each process that is running in the background (unless you start more than one process in a pipeline, in which case the shell reports the number of the last process). You can also use the **ps** command to find the process ID number of commands.
kill -l [*ExitStatus*]	Lists signal names. For more information, see the **kill** command.
let *Expression* ...	Evaluates specified arithmetic expressions. The exit status is 0 if the value of the last expression is nonzero, and 1 otherwise. See "Arithmetic evaluation in the Korn shell or POSIX shell" on page 210 for more information.
print [**-Rnprsu** [*n*]] [*Argument* ...]	Prints shell output. If you do not specify any flags, or if you specify the hyphen (-) or double hyphen (–) flags, the arguments are printed to standard output as described by the **echo** command. The flags do the following: **-R** Prints in raw mode (the escape conventions of the **echo** command are ignored). The **-R** flag prints all subsequent arguments and flags other than **-n**. **-n** Prevents a newline character from being added to the output. **-p** Writes the arguments to the pipe of the process run with |& instead of to standard output. **-r** Prints in raw mode. The escape conventions of the **echo** command are ignored. **-s** Writes the arguments to the history file instead of to standard output. **-u** Specifies a one-digit file descriptor unit number, *n*, on which the output is placed. The default is 1.
pwd	Equivalent to `print -r - $PWD`. **Note:** The internal Korn shell **pwd** command does not support symbolic links.
read [**-prsu** [**n**]] [*Name?Prompt*] [*Name...*]	Takes shell input. One line is read and broken up into fields, using the characters in the IFS variable as separators. For more information, see the **read** command.
setgroups	Executes the **/usr/bin/setgroups** command, which runs as a separate shell. See the **setgroups** command for information on this command. There is one difference, however. The **setgroups** built-in command invokes a subshell, but the **setgroups** command replaces the currently executing shell. Because the built-in command is supported only for compatibility, it is recommended that scripts use the absolute path name /usr/bin/setgroups rather than the shell built-in command.
setsenv	Executes the /usr/bin/setsenv command, which replaces the currently executing shell. See the **setsenv** command for information on this command.

Item	Description
test	Same as [*expression*]. See "Conditional expressions for the Korn shell or POSIX shell" on page 214 for usage and description.

ulimit [**-HSacdfmst**] [*Limit*]	Sets or displays user-process resource limits as defined in the `/etc/security/limits` file. This file contains the following default limits:
	`fsize = 2097151`
	`core = 2048`
	`cpu = 3600`
	`data = 131072`
	`rss = 65536`
	`stack = 8192`
	`threads = -1`
	These values are used as default settings when a user is added to the system. The values are set with the **mkuser** command when the user is added to the system or changed with the **chuser** command.
	Limits are categorized as either soft or hard. Users might change their soft limits, up to the maximum set by the hard limits, with the **ulimit** command. You must have root user authority to change resource hard limits.
	Many systems do not contain one or more of these limits. The limit for a specified resource is set when the *Limit* parameter is specified. The value of the *Limit* parameter can be a number in the unit specified with each resource or the value `unlimited`. You can specify the following **ulimit** command flags:
	-H
	Specifies that the hard limit for the given resource is set. If you have root user authority, you can increase the hard limit. Any user can decrease it.
	-S
	Specifies that the soft limit for the given resource is set. A soft limit can be increased up to the value of the hard limit. If neither the **-H** or **-S** options are specified, the limit applies to both.
	-a
	Lists all of the current resource limits.
	-c
	Specifies the number of 512-byte blocks on the size of core dumps.
	-d
	Specifies the size, in KB, of the data area.
	-f
	Specifies the number of 512-byte blocks for files written by child processes (files of any size can be read).
	-m
	Specifies the number of KB for the size of physical memory.
	-n
	Specifies the limit on the number of file descriptors a process might have open.
	-r
	Specifies the limit on the number of threads per process.
	-s
	Specifies the number of KB for the size of the stack area.
	-t
	Specifies the number of seconds to be used by each process.

Item	Description
	The current resource limit is printed when you omit the *Limit* variable. The soft limit is printed unless you specify the **-H** flag. When you specify more than one resource, the limit name and unit is printed before the value. If no option is given, the **-f** flag is assumed. When you change the value, set both hard and soft limits to *Limit* unless you specify **-H** or **-S**. For more information about user and system resource limits, see the `getrlimit, setrlimit, or vlimit` subroutine.
umask [**-S**] [*Mask*]	Determines file permissions. This value, along with the permissions of the creating process, determines a file's permissions when the file is created. The default is 022. If the *Mask* parameter is not specified, the **umask** command displays to standard output the file-mode creation mask of the current shell environment. For more information about file permissions, see the **umask** command.
unalias { **-a** | *AliasName...* }	Removes the definition for each alias name specified, or removes all alias definitions if the **-a** flag is used. Alias definitions are removed from the current shell environment. For more information, see the **unalias** command.
wait [*ProcessID...*]	Waits for the specified job and terminates. If you do not specify a job, the command waits for all currently active child processes. The exit status from this command is that of the process for which it waits. For more information, see the **wait** command.
whence [**-pv**] *Name ...*	Indicates, for each name specified, how it would be interpreted if used as a command name. When used without either flag, **whence** will display the absolute path name, if any, that corresponds to each name. **-p** Performs a path search for the specified name or names even if these are aliases, functions, or reserved words. **-v** Produces a more verbose report that specifies the type of each name.

Job control in the Korn shell or POSIX shell

The Korn shell, or POSIX shell, provides a facility to control command sequences, or *jobs*.

When you execute the **set -m** special command, the Korn shell associates a job with each pipeline. It keeps a table of current jobs, printed by the **jobs** command, and assigns them small integer numbers.

When a job is started in the background with an ampersand (&), the shell prints a line that looks like the following:

```
[1] 1234
```

This output indicates that the job, which was started in the background, was job number 1. It also shows that the job had one (top-level) process with a process ID of 1234.

If you are running a job and want to do something else, use the Ctrl-Z key sequence. This key sequence sends a **STOP** signal to the current job. The shell normally indicates that the job has been stopped and then displays a shell prompt. You can then manipulate the state of this job (putting it in the background with the **bg** command), run other commands, and then eventually return the job to the foreground with the **fg** command. The Ctrl-Z key sequence takes effect immediately, and is like an interrupt in that the shell discards pending output and unread input when you type the sequence.

A job being run in the background stops if it tries to read from the terminal. Background jobs are normally allowed to produce output. You can disable this option by issuing the **stty tostop** command. If you set this terminal option, then background jobs stop when they try to produce output or read input.

You can refer to jobs in the Korn shell in several ways. A job is referenced by the process ID of any of its processes or in one of the following ways:

Item	Description
%*Number*	Specifies the job with the given number.
%*String*	Specifies any job whose command line begins with the *String* variable.
%?*String*	Specifies any job whose command line contains the *String* variable.
%%	Specifies the current job.
%+	Equivalent to %%.
%-	Specifies the previous job.

This shell immediately recognizes changes in the process state. It normally informs you whenever a job becomes blocked so that no further progress is possible. The shell does this just before it prints a prompt so that it does not otherwise disturb your work.

When the monitor mode is on, each completed background job triggers traps set for the **CHLD** signal.

If you try to leave the shell (either by typing `exit` or using the Ctrl-D key sequence) while jobs are stopped or running, the system warns you with the message `There are stopped (running) jobs.` Use the **jobs** command to see which jobs are affected. If you immediately try to exit again, the shell terminates the stopped and running jobs without warning.

Signal handling
The `SIGINT` and `SIGQUIT` signals for an invoked command are ignored if the command is followed by an ampersand (&) and the job **monitor** option is not active. Otherwise, signals have the values that the shell inherits from its parent.

When a signal for which a trap has been set is received while the shell is waiting for the completion of a foreground command, the trap associated with that signal will not be executed until after the foreground command has completed. Therefore, a trap on a `CHILD` signal is not performed until the foreground job terminates.

Inline editing in the Korn shell or POSIX shell

Normally, you type each command line from a terminal device and follow it by a newline character (**RETURN** or **LINE FEED**). When you activate the emacs, gmacs, or vi inline editing option, you can edit the command line.

The following commands enter edit modes:

Item	Description
set -o emacs	Enters emacs editing mode and initiates an emacs-style inline editor.
set -o gmacs	Enters emacs editing mode and initiates a gmacs-style inline editor.
set -o vi	Enters vi editing mode and initiates a vi-style inline editor.

An editing option is automatically selected each time the *VISUAL* or *EDITOR* variable is assigned a value that ends in any of these option names.

Note: To use the editing features, your terminal must accept **RETURN** as a carriage return without line feed. A space must overwrite the current character on the screen.

Each editing mode opens a window at the current line. The window width is the value of the *COLUMNS* variable if it is defined; otherwise, the width is 80 character spaces. If the line is longer than the window width minus two, the system notifies you by displaying a mark at the end of the window. As

the cursor moves and reaches the window boundaries, the window is centered about the cursor. The marks displayed are as follows:

Item	Description
>	Indicates that the line extends on the right side of the window.
<	Indicates that the line extends on the left side of the window.
*	Indicates that the line extends on both sides of the window.

The search commands in each edit mode provide access to the Korn shell history file. Only strings are matched. If the leading character in the string is a carat (^), the match must begin at the first character in the line.

Related concepts

Korn shell or POSIX shell commands

The Korn shell is an interactive command interpreter and command programming language. It conforms to the Portable Operating System Interface for Computer Environments (POSIX), an international standard for operating systems.

emacs editing mode

The emacs editing mode is entered when you enable either the **emacs** or **gmacs** option. The only difference between these two modes is the way each handles the Ctrl-T edit command.

To edit, move the cursor to the point needing correction and insert or delete characters or words, as needed. All of the editing commands are control characters or escape sequences.

Edit commands operate from any place on a line (not only at the beginning). Do not press the Enter key or line-feed (Down Arrow) key after edit commands, except as noted.

Item	Description
Ctrl-F	Moves the cursor forward (right) one character.
Esc-F	Moves the cursor forward one word (a string of characters consisting of only letters, digits, and underscores).
Ctrl-B	Moves the cursor backward (left) one character.
Esc-B	Moves the cursor backward one word.
Ctrl-A	Moves the cursor to the beginning of the line.
Ctrl-E	Moves the cursor to the end of the line.
Ctrl-] *c*	Moves the cursor forward on the current line to the indicated character.
Esc-Ctrl-] *c*	Moves the cursor backward on the current line to the indicated character.
Ctrl-X Ctrl-X	Interchanges the cursor and the mark.
ERASE	Deletes the previous character. (User-defined erase character as defined by the **stty** command, usually the Ctrl-H key sequence.)
Ctrl-D	Deletes the current character.
Esc-D	Deletes the current word.
Esc-Backspace	Deletes the previous word.
Esc-H	Deletes the previous word.
Esc-Delete	Deletes the previous word. If your interrupt character is the Delete key, this command does not work.

Item	Description
Ctrl-T	Transposes the current character with the next character in emacs mode. Transposes the two previous characters in gmacs mode.
Ctrl-C	Capitalizes the current character.
Esc-C	Capitalizes the current word.
Esc-L	Changes the current word to lowercase.
Ctrl-K	Deletes from the cursor to the end of the line. If preceded by a numeric parameter whose value is less than the current cursor position, this editing command deletes from the given position up to the cursor. If preceded by a numeric parameter whose value is greater than the current cursor position, this editing command deletes from the cursor up to the given cursor position.
Ctrl-W	Deletes from the cursor to the mark.
Esc-P	Pushes the region from the cursor to the mark on the stack.
KILL	User-defined kill character as defined by the **stty** command, usually the Ctrl-G key sequence or @. Kills the entire current line. If two kill characters are entered in succession, all subsequent kill characters cause a line feed (useful when using paper terminals).
Ctrl-Y	Restores the last item removed from the line. (Yanks the item back to the line.)
Ctrl-L	Line feeds and prints the current line.
Ctrl-@	(Null character) Sets a mark.
Esc-space	Sets a mark.
Ctrl-J	(New line) Executes the current line.
Ctrl-M	(Return) Executes the current line.
EOF	Processes the end-of-file character, normally the Ctrl-D key sequence, as an end-of-file only if the current line is null.
Ctrl-P	Fetches the previous command. Each time the Ctrl-P key sequence is entered, the previous command back in time is accessed. Moves back one line when not on the first line of a multiple-line command.
Esc-<	Fetches the least recent (oldest) history line.
Esc->	Fetches the most recent (youngest) history line.
Ctrl-N	Fetches the next command line. Each time the Ctrl-N key sequence is entered, the next command line forward in time is accessed.
Ctrl-R *String*	Reverses search history for a previous command line containing the string specified by the **String** parameter. If a value of 0 is given, the search is forward. The specified string is terminated by an Enter or newline character. If the string is preceded by a carat (^), the matched line must begin with the **String** parameter. If the **String** parameter is omitted, then the next command line containing the most recent **String** parameter is accessed. In this case, a value of 0 reverses the direction of the search.
Ctrl-O	(Operate) Executes the current line and fetches the next line relative to the current line from the history file.

Item	Description
Esc *Digits*	(Escape) Defines the numeric parameter. The digits are taken as a parameter to the next command. The commands that accept a parameter are **Ctrl-F**, **Ctrl-B**, **ERASE**, **Ctrl-C**, **Ctrl-D**, **Ctrl-K**, **Ctrl-R**, **Ctrl-P**, **Ctrl-N**, **Ctrl-]**, **Esc-.**, **Esc-Ctrl-]**, **Esc-_**, **Esc-B**, **Esc-C**, **Esc-D**, **Esc-F**, **Esc-H**, **Esc-L**, and **Esc-Ctrl-H**.
Esc *Letter*	(Soft-key) Searches the alias list for an alias named *_Letter.* If an alias of this name is defined, its value is placed into the input queue. The *Letter* parameter must not specify one of the escape functions.
Esc-[*Letter*	(Soft-key) Searches the alias list for an alias named double underscore Letter (*__Letter*). If an alias of this name is defined, its value is placed into the input queue. This command can be used to program function keys on many terminals.
Esc-.	Inserts on the line the last word of the previous command. If preceded by a numeric parameter, the value of this parameter determines which word to insert rather than the last word.
Esc-_	Same as the **Esc-**. key sequence.
Esc-*	Attempts file name substitution on the current word. An asterisk (*) is appended if the word does not match any file or contain any special pattern characters.
Esc-Esc	File name completion. Replaces the current word with the longest common prefix of all file names that match the current word with an asterisk appended. If the match is unique, a slash (/) is appended if the file is a directory and a space is appended if the file is not a directory.
Esc-=	Lists the files that match the current word pattern as if an asterisk (*) were appended.
Ctrl-U	Multiplies the parameter of the next command by 4.
****	Escapes the next character. Editing characters and the **ERASE**, **KILL** and **INTERRUPT** (normally the Delete key) characters can be entered in a command line or in a search string if preceded by a backslash (\). The backslash removes the next character's editing features, if any.
Ctrl-V	Displays the version of the shell.
Esc-#	Inserts a pound sign (#) at the beginning of the line and then executes the line. This causes a comment to be inserted in the history file.

vi editing mode
The vi editing mode has two typing modes.

The modes are:

- **Input mode.** When you enter a command, the vi editor is in input mode.
- **Control mode.** Press the Esc key to enter control mode.

Most control commands accept an optional repeat **Count** parameter prior to the command. When in vi mode on most systems, canonical processing is initially enabled. The command is echoed again if one or more of the following are true:

- The speed is 1200 baud or greater.
- The command contains any control characters.
- Less than one second has elapsed since the prompt was printed.

The Esc character terminates canonical processing for the remainder of the command, and you can then modify the command line. This scheme has the advantages of canonical processing with the type-ahead echoing of raw mode. If the **viraw** option is also set, canonical processing is always disabled. This mode

is implicit for systems that do not support two alternate end-of-line delimiters and might be helpful for certain terminals.

Available vi edit commands are grouped into categories. The categories are as follows:

Input edit commands
The input edit commands for the Korn shell are described below.

Note: By default, the editor is in input mode.

Item	Description
ERASE	Deletes the previous character. (User-defined erase character as defined by the **stty** command, usually Ctrl-H or #.)
Ctrl-W	Deletes the previous blank separated word.
Ctrl-D	Terminates the shell.
Ctrl-V	Escapes the next character. Editing characters, such as the **ERASE** or **KILL** characters, can be entered in a command line or in a search string if preceded by a Ctrl-V key sequence. The Ctrl-V key sequence removes the next character's editing features (if any).
****	Escapes the next **ERASE** or **KILL** character.

Motion edit commands
The motion edit commands for the Korn shell are described below.

Motion edit commands move the cursor as follows:

Item	Description
[*Count*]**l**	Moves the cursor forward (right) one character.
[*Count*]**w**	Moves the cursor forward one alphanumeric word.
[*Count*]**W**	Moves the cursor to the beginning of the next word that follows a blank.
[*Count*]**e**	Moves the cursor to the end of the current word.
[*Count*]**E**	Moves the cursor to the end of the current blank-separated word.
[*Count*]**h**	Moves the cursor backward (left) one character.
[*Count*]**b**	Moves the cursor backward one word.
[*Count*]**B**	Moves the cursor to the previous blank-separated word.
[*Count*]**|**	Moves the cursor to the column specified by the *Count* parameter.
[*Count*]**f***c*	Finds the next character *c* in the current line.
[*Count*]**F***c*	Finds the previous character *c* in the current line.
[*Count*]**t***c*	Equivalent to **f** followed by **h**.
[*Count*]**T***c*	Equivalent to **F** followed by **l**.
[*Count*]**;**	Repeats for the number of times specified by the *Count* parameter the last single-character find command: **f**, **F**, **t**, or **T**.
[*Count*]**,**	Reverses the last single-character find command the number of times specified by the *Count* parameter.
0	Moves the cursor to the start of a line.
^	Moves the cursor to the first nonblank character in a line.
$	Moves the cursor to the end of a line.

Search edit commands
Search edit commands access your command history as follows:

Item	Description
[*Count*]**k**	Fetches the previous command.
[*Count*]**-**	Equivalent to the **k** command.
[*Count*]**j**	Fetches the next command. Each time the **j** command is entered, the next command is accessed.
[*Count*]**+**	Equivalent to the **j** command.
[*Count*]**G**	Fetches the command whose number is specified by the *Count* parameter. The default is the least recent history command.
/*String*	Searches backward through history for a previous command containing the specified string. The string is terminated by a RETURN or newline character. If the specified string is preceded by a carat (^), the matched line must begin with the *String* parameter. If the value of the *String* parameter is null, the previous string is used.
?*String*	Same as **/***String* except that the search is in the forward direction.
n	Searches for the next match of the last pattern to **/***String* or **?** commands.
N	Searches for the next match of the last pattern to **/***String* or **?** commands, but in the opposite direction. Searches history for the string entered by the previous **/***String* command.

Text modification edit commands
Text-modification edit commands modify the line as follows:

Item	Description
a	Enters the input mode and enters text after the current character.
A	Appends text to the end of the line. Equivalent to the **$a** command.
[*Count*]**c***Motion* **c**[*Count*]*Motion*	Deletes the current character through the character to which the *Motion* parameter specifies to move the cursor, and enters input mode. If the value of the *Motion* parameter is **c**, the entire line is deleted and the input mode is entered.
C	Deletes the current character through the end of the line and enters input mode. Equivalent to the **c$** command.
S	Equivalent to the **cc** command.
D	Deletes the current character through the end of line. Equivalent to the **d$** command.

Item	Description
[*Count*]**d***Motion* **d**[*Count*]*Motion*	Deletes the current character up to and including the character specified by the *Motion* parameter. If *Motion* is **d**, the entire line is deleted.
i	Enters the input mode and inserts text before the current character.
I	Inserts text before the beginning of the line. Equivalent to the **0i** command.
[*Count*]**P**	Places the previous text modification before the cursor.

Item	Description
[*Count*]**p**	Places the previous text modification after the cursor.
R	Enters the input mode and types over the characters on the screen.
[*Count*]**r***c*	Replaces the number of characters specified by the *Count* parameter, starting at the current cursor position, with the characters specified by the *c* parameter. This command also advances the cursor after the characters are replaced.
[*Count*]**x**	Deletes the current character.
[*Count*]**X**	Deletes the preceding character.
[*Count*]**.**	Repeats the previous text-modification command.
[*Count*]~	Inverts the case of the number of characters specified by the *Count* parameter, starting at the current cursor position, and advances the cursor.
[*Count*]_	Appends the word specified by the *Count* parameter of the previous command and enters input mode. The last word is used if the *Count* parameter is omitted.
*****	Appends an asterisk (*) to the current word and attempts file name substitution. If no match is found, it rings the bell. Otherwise, the word is replaced by the matching pattern and input mode is entered.
****	File name completion. Replaces the current word with the longest common prefix of all file names matching the current word with an asterisk (*) appended. If the match is unique, a slash / is appended if the file is a directory. A space is appended if the file is not a directory.

Miscellaneous edit commands
The following edit commands are used commonly.

Item	Description
[*Count*]**y***Motion*	
y[*Count*]*Motion*	Yanks the current character up to and including the character marked by the cursor position specified by the *Motion* parameter and puts all of these characters into the delete buffer. The text and cursor are unchanged.
Y	Yanks from the current position to the end of the line. Equivalent to the **y$** command.
u	Undoes the last text-modifying command.
U	Undoes all the text-modifying commands performed on the line.
[*Count*]**v**	Returns the command `fc -e ${VISUAL:-${EDITOR:-vi}}` *Count* in the input buffer. If the *Count* parameter is omitted, then the current line is used.
Ctrl-L	Line feeds and prints the current line. This command is effective only in control mode.
Ctrl-J	(New line) Executes the current line regardless of the mode.
Ctrl-M	(Return) Executes the current line regardless of the mode.

Item	Description
#	Sends the line after inserting a pound sign (*#*) in front of the line. Useful if you want to insert the current line in the history without executing it. If the command line contains a pipe or semicolon or newline character, then additional pound signs (*#*) will be inserted in front of each of these symbols. To delete all pound signs, retrieve the command line from history and enter another pound sign (*#*).
=	Lists the file names that match the current word as if an asterisk were appended to it.
@*Letter*	Searches the alias list for an alias named *_Letter*. If an alias of this name is defined, its value is placed into the input queue for processing.

Korn shell or POSIX shell commands

The Korn shell is an interactive command interpreter and command programming language. It conforms to the Portable Operating System Interface for Computer Environments (POSIX), an international standard for operating systems.

POSIX is not an operating system, but is a *standard* aimed at portability of applications, at the source level, across many systems. POSIX features are built on top of the Korn shell. The Korn shell (also known as the POSIX shell) offers many of the same features as the Bourne and C shells, such as I/O redirection capabilities, variable substitution, and file name substitution. It also includes several additional command and programming language features:

Note: There is a restricted version of Korn shell available, called `rksh`. For more details, refer to the **rksh** command.

Item	Description
Arithmetic evaluation	The Korn shell, or POSIX shell, can perform integer arithmetic using the built-in **let** command, using any base from 2 to 36. In order to enable recognition of numbers starting with 0 (Octal) and 0x (Hexadecimal) in the Korn shell, run the following commands: **export XPG_SUS_ENV=ON** Exporting the XPG_SUS_ENV variable causes the commands that are run and the libraries that they use to be completely POSIX-compliant. **Note:** Because the entire library system becomes POSIX-compliant, a given command's default expected behavior might change. **export OCTAL_CONST=ON** Exporting this variable causes the interpretation of constants declared in the Korn shell to be POSIX-compliant as far as the recognition of octal and hexadecimal constants is concerned.
Command history	The Korn shell, or POSIX shell, stores a file that records all of the commands you enter. You can use a text editor to alter a command in this history file and then reissue the command.
Coprocess facility	Enables you to run programs in the background and send and receive information to these background processes.
Editing	The Korn shell, or POSIX shell, offers inline editing options that enable you to edit the command line. Editors similar to emacs, gmacs, and vi are available.

A Korn shell command is one of the following:

- Simple command
- Pipeline

- List
- Compound command
- Function

When you issue a command in the Korn shell or POSIX shell, the shell evaluates the command and does the following:

- Makes all indicated substitutions.
- Determines whether the command contains a slash (/). If it does, the shell runs the program named by the specified path name.

If the command does not contain a slash (/), the Korn shell or POSIX shell continues with the following actions:

- Determines whether the command is a special built-in command. If it is, the shell runs the command within the current shell process.
- Compares the command to user-defined functions. If the command matches a user-defined function, then the positional parameters are saved and then reset to the arguments of the *function* call. When the function completes or issues a return, the positional parameter list is restored, and any trap set on EXIT within the function is carried out. The value of a function is the value of the last command executed. A function is carried out in the current shell process.
- If the command name matches the name of a regular built-in command, then that regular built-in command will be invoked.
- Creates a process and attempts to carry out the command by using the **exec** command (if the command is neither a built-in command nor a user-defined function).

The Korn shell, or POSIX shell, searches each directory in a specified path for an executable file. The *PATH* shell variable defines the search path for the directory containing the command. Alternative directory names are separated with a colon (`:`). The default path is `/usr/bin:` (specifying the `/usr/bin` directory, and the current directory, in that order). The current directory is specified by two or more adjacent colons, or by a colon at the beginning or end of the path list.

If the file has execute permission but is not a directory or an a `.out` file, the shell assumes that it contains shell commands. The current shell process creates a subshell to read the file. All nonexported aliases, functions, and named parameters are removed from the file. If the shell command file has *read* permission, or if the **setuid** or **setgid** bits are set on the file, then the shell runs an agent that sets up the permissions and carries out the shell with the shell command file passed down as an open file. A parenthesized command is run in a subshell without removing nonexported quantities.

Related concepts

Available shells
The following are the shells that are provided with AIX.

Coprocess facility
The Korn shell, or POSIX shell, allows you to run one or more commands as background processes. These commands, run from within a shell script, are called *coprocesses*.

Inline editing in the Korn shell or POSIX shell
Normally, you type each command line from a terminal device and follow it by a newline character (**RETURN** or **LINE FEED**). When you activate the emacs, gmacs, or vi inline editing option, you can edit the command line.

Arithmetic evaluation in the Korn shell or POSIX shell
The Korn shell or POSIX shell regular built-in **`let`** command enables you to perform integer arithmetic.

Korn shell or POSIX shell built-in commands

Special commands are built in to the Korn shell and POSIX shell and executed in the shell process.

Korn shell compound commands
A compound command can be a list of simple commands or a pipeline, or it can begin with a reserved word. Most of the time, you will use compound commands such as **if**, **while**, and **for** when you are writing shell scripts.

The following is a list of list of Korn shell or POSIX shell compound commands:

Command syntax	Description
for *Identifier* [**in** *Word* ...] **;do** *List* **;done**	Each time a **for** command is executed, the **Identifier** parameter is set to the next word taken from the **in** *Word* ... list. If the **in** *Word* ... command is omitted, then the **for** command executes the **do** *List* command once for each positional parameter that is set. Execution ends when there are no more words in the list.
select *Identifier* [**in** *Word* ...] **;do** *List* **;done**	A **select** command prints on standard error (file descriptor 2) the set of words specified, each preceded by a number. If the **in** *Word* ... command is omitted, then the positional parameters are used instead. The **PS3** prompt is printed and a line is read from the standard input. If this line consists of the number of one of the listed words, then the value of the *Identifier* parameter is set to the word corresponding to this number. If the line read from standard input is empty, the selection list is printed again. Otherwise, the value of the *Identifier* parameter is set to null. The contents of the line read from standard input is saved in the **REPLY** parameter. The *List* parameter is executed for each selection until a break or an end-of-file character is encountered.
case *Word* **in** [[**(**] *Pattern* [| *Pattern*] ... **)** *List* **;;**] ... **esac**	A **case** command executes the *List* parameter associated with the first *Pattern* parameter that matches the *Word* parameter. The form of the patterns is the same as that used for file name substitution.
if *List* **;then** *List* [**elif** *List* **;then** *List*] ... [**;else** *List*] **;fi**	The *List* parameter specifies a list of commands to be run. The shell executes the **if** *List* command first. If a zero exit status is returned, it executes the **then** *List* command. Otherwise, the commands specified by the *List* parameter following the **elif** command are executed. If the value returned by the last command in the **elif** *List* command is zero, the **then** *List* command is executed. If the value returned by the last command in the **then** *List* command is zero, the **else** *List* command is executed. If no commands specified by the *List* parameters are executed for the **else** or **then** command, the **if** command returns a zero exit status.
while *List* **;do** *List* **;done** **until** *List* **;do** *List* **;done**	The *List* parameter specifies a list of commands to be run. The **while** command repeatedly executes the commands specified by the *List* parameter. If the exit status of the last command in the **while** *List* command is zero, the **do** *List* command is executed. If the exit status of the last command in the **while** *List* command is not zero, the loop terminates. If no commands in the **do** *List* command are executed, then the **while** command returns a zero exit status. The **until** command might be used in place of the **while** command to negate the loop termination test.
(*List***)**	The *List* parameter specifies a list of commands to run. The shell executes the *List* parameter in a separate environment. **Note:** If two adjacent open parentheses are needed for nesting, you must insert a space between them to differentiate between the command and arithmetic evaluation.

Command syntax	Description
{ *List***;}**	The *List* parameter specifies a list of commands to run. The *List* parameter is simply executed. **Note:** Unlike the metacharacters `( )`, `{ }` denote reserved words (used for special purposes, not as user-declared identifiers). To be recognized, these reserved words must appear at the beginning of a line or after a semicolon (`;`).
[[*Expression*]]	Evaluates the *Expression* parameter. If the expression is true, then the command returns a zero exit status.
function *Identifier* **{** *List* **;}** or **function** *Identifier* **() {***List* **;}**	Defines a function that is referred to by the *Identifier* parameter. The body of the function is the specified list of commands enclosed by `{ }`. The `( )` consists of two operators, so mixing blank characters with the *identifier*, `(` and `)` is permitted, but is not necessary.
time *Pipeline*	Executes the *Pipeline* parameter. The elapsed time, user time, and system time are printed to standard error.

Related concepts

Parameters in the Korn shell
Korn shell parameters are discussed below.

Shell startup
You can start the Korn shell with the **ksh** command, **psh** command (POSIX shell), or the **exec** command.

If the shell is started by the **exec** command, and the first character of zero argument (**$0**) is the hyphen (`-`), then the shell is assumed to be a login shell. The shell first reads commands from the `/etc/profile` file and then from either the `.profile` file in the current directory or from the `$HOME/.profile` file, if either file exists. Next, the shell reads commands from the file named by performing parameter substitution on the value of the ENV environment variable, if the file exists.

If you specify the *File* [*Parameter*] parameter when invoking the Korn shell or POSIX shell, the shell runs the script file identified by the *File* parameter, including any parameters specified. The script file specified must have read permission; any **setuid** and **setgid** settings are ignored. The shell then reads the commands.

> **Note:** Do not specify a script file with the **-c** or **-s** flags when invoking the Korn shell or POSIX shell.

For more information on positional parameters, see "Parameters in the Korn shell" on page 224.

Related concepts

Parameters in the Korn shell
Korn shell parameters are discussed below.

Korn shell environment
All variables (with their associated values) known to a command at the beginning of its execution constitute its *environment*.

This environment includes variables that a command inherits from its parent process and variables specified as keyword parameters on the command line that calls the command. The shell interacts with the environment in several ways. When it is started, the shell scans the environment and creates a parameter for each name found, giving the parameter the corresponding value and marking it for export. Executed commands inherit the environment.

If you modify the values of the shell parameters or create new ones using the **export** or **typeset -x** commands, the parameters become part of the environment. The environment seen by any executed command is therefore composed of any name-value pairs originally inherited by the shell, whose values might be modified by the current shell, plus any additions that resulted from using the **export** or **typeset -x** commands. The executed command (subshell) will see any modifications it makes to the

environment variables it has inherited, but for its child shells or processes to see the modified values, the subshell must export these variables.

The environment for any simple command or function is changed by prefixing with one or more parameter assignments. A parameter assignment argument is a word of the form *Identifier*=*Value*. Thus, the two following expressions are equivalent (as far as the execution of the command is concerned):

```
TERM=450 Command arguments
```

```
(export TERM; TERM=450; Command arguments)
```

Korn shell functions
The **function** reserved word defines shell functions. The shell reads and stores functions internally. Alias names are resolved when the function is read. The shell executes functions in the same manner as commands, with the arguments passed as positional parameters.

The Korn shell or POSIX shell executes functions in the environment from which functions are invoked. All of the following are shared by the function and the invoking script, and side effects can be produced:

- Variable values and attributes (unless you use **typeset** command within the function to declare a local variable)
- Working directory
- Aliases, function definitions, and attributes
- Special parameter $
- Open files

The following are not shared between the function and the invoking script, and there are no side effects:

- Positional parameters
- Special parameter #
- Variables in a variable assignment list when the function is invoked
- Variables declared using **typeset** command within the function
- Options
- Traps. However, signals ignored by the invoking script will also be ignored by the function.

Note: In earlier versions of the Korn shell, traps other than **EXIT** and **ERR** were shared by the function as well as the invoking script.

If trap on **0** or **EXIT** is executed *inside* the body of a function, then the action is executed after the function completes, in the environment that called the function. If the trap is executed *outside* the body of a function, then the action is executed upon exit from the Korn shell. In earlier versions of the Korn shell, no trap on **0** or **EXIT** outside the body of a function was executed upon exit from the function.

When a function is executed, it has the same syntax-error and variable-assignment properties described in Korn shell or POSIX shell built-in commands.

The compound command is executed whenever the function name is specified as the name of a simple command. The operands to the command temporarily will become the positional parameters during the execution of the compound command. The special parameter # will also change to reflect the number of operands. The special parameter 0 will not change.

The **return** special command is used to return from function calls. Errors within functions return control to the caller.

Function identifiers are listed with the **-f** or **+f** option of the **typeset** special command. The **-f** option also lists the text of functions. Functions are undefined with the **-f** option of the **unset** special command.

Ordinarily, functions are unset when the shell executes a shell script. The **-xf** option of the **typeset** special command allows a function to be exported to scripts that are executed without a separate invocation of

the shell. Functions that must be defined across separate invocations of the shell should be specified in the ENV file with the **-xf** option of the **typeset** special command.

The exit status of a function definition is zero if the function was not successfully declared. Otherwise, it will be greater than zero. The exit status of a function invocation is the exit status of the most recent command executed by the function.

Related concepts

Parameters in the Korn shell
Korn shell parameters are discussed below.

Korn shell or POSIX shell built-in commands
Special commands are built in to the Korn shell and POSIX shell and executed in the shell process.

Korn shell or POSIX shell command history
The Korn shell or POSIX shell saves commands entered from your terminal device to a history file.

If set, the *HISTFILE* variable value is the name of the history file. If the *HISTFILE* variable is not set or cannot be written, the history file used is `$HOME/.sh_history`. If the history file does not exist and the Korn shell cannot create it, or if it does exist and the Korn shell does not have permission to append to it, then the Korn shell uses a temporary file as the history file. The shell accesses the commands of all interactive shells using the same named history file with appropriate permissions.

By default, the Korn shell or POSIX shell saves the text of the last 128 commands for nonroot users and 512 commands for the root user. The history file size (specified by the *HISTSIZE* variable) is not limited, although a very large history file can cause the Korn shell to start slowly.

Command history substitution
Use the **fc** built-in command to list or edit portions of the history file. To select a portion of the file to edit or list, specify the number or the first character or characters of the command.

You can specify a single command or range of commands.

If you do not specify an editor program as an argument to the **fc** regular built-in command, the editor specified by the *FCEDIT* variable is used. If the *FCEDIT* variable is not defined, then the `/usr/bin/ed` file is used. The edited command or commands are printed and run when you exit the editor.

The editor name hyphen (`-`) is used to skip the editing phase and run the command again. In this case, a substitution parameter of the form `Old=New` can be used to modify the command before it is run. For example, if `r` is aliased to `fc -e -`, then typing `r bad=good c` runs the most recent command that starts with the letter `c` and replaces the first occurrence of the `bad` string with the `good` string.

Related tasks

Listing previously entered commands (history command)
Use the **history** command to list commands that you have previously entered.

Command aliasing in the Korn shell or POSIX shell

The Korn shell, or POSIX shell, allows you to create aliases to customize commands.

The **alias** command defines a word of the form `Name=String` as an alias. When you use an alias as the first word of a command line, the Korn shell checks to see if it is already processing an alias with the same name. If it is, the Korn shell does not replace the alias name. If an alias with the same name is not already being processed, the Korn shell replaces the alias name by the value of the alias.

The first character of an alias name can be any printable character except the metacharacters. The remaining characters must be the same as for a valid identifier. The replacement string can contain any valid shell text, including the metacharacters.

If the last character of the alias value is a blank, the shell also checks the word following the alias for alias substitution. You can use aliases to redefine special built-in commands but not to redefine reserved words. Alias definitions are not inherited across invocations of **ksh**. However, if you specify **alias -x**, the alias stays in effect for scripts invoked by name that do not invoke a separate shell. To export an alias definition and to cause child processes to have access to them, you must specify **alias -x** and the alias definition in your environment file.

Use the **alias** command to create, list, and export aliases.

Use the **unalias** command to remove aliases.

The format for creating an alias is as follows:

```
alias Name=String
```

where the **Name** parameter specifies the name of the alias, and the **String** parameter specifies the value of the alias.

The following exported aliases are predefined by the Korn shell but can be unset or redefined. It is not recommended that you change them, because this might later confuse anyone who expects the alias to work as predefined by the Korn shell.

```
autoload='typeset -fu'
false='let 0'
functions='typeset -f'
hash='alias -t'
history='fc -l'
integer='typeset -i'
nohup='nohup '
r='fc -e -'
true=':'
type='whence -v'
```

Aliases are not supported on noninteractive invocations of the Korn shell (**ksh**); for example, in a shell script, or with the **-c** option in **ksh**, as in the following:

```
ksh -c alias
```

Related tasks

Creating a command alias (alias shell command)

An *alias* lets you create a shortcut name for a command, file name, or any shell text. By using aliases, you save a lot of time when doing tasks you do frequently. You can create a command alias.

Tracked aliases

Frequently, aliases are used as shorthand for full path names. One aliasing facility option allows you to automatically set the value of an alias to the full path name of a corresponding command. This special type of alias is a *tracked* alias.

Tracked aliases speed execution by eliminating the need for the shell to search the *PATH* variable for a full path name.

The **set -h** command turns on command *tracking* so that each time a command is referenced, the shell defines the value of a tracked alias. This value is undefined each time you reset the *PATH* variable.

These aliases remain tracked so that the next subsequent reference will redefine the value. Several tracked aliases are compiled into the shell.

Tilde substitution

After the shell performs alias substitution, it checks each word to see if it begins with an unquoted tilde (~). If it does, the shell checks the word, up to the first slash (/), to see if it matches a user name in the `/etc/passwd` file. If the shell finds a match, it replaces the ~ character and the name with the login directory of the matched user. This process is called *tilde substitution*.

The shell does not change the original text if it does not find a match. The Korn shell also makes special replacements if the ~ character is the only character in the word or followed by plus sign (+) or hyphen (-):

Item	Description
~	Replaced by the value of the *HOME* variable
~+	Replaced by the *$PWD* variable (the full path name of the current directory)
~-	Replaced by the *$OLDPWD* variable (the full path name of the previous directory)

In addition, the shell attempts tilde substitution when the value of a variable assignment parameter begins with a tilde ~ character.

Bourne shell

The Bourne shell is an interactive command interpreter and command programming language.

The **bsh** command runs the Bourne shell.

The Bourne shell can be run either as a login shell or as a subshell under the login shell. Only the **login** command can call the Bourne shell as a login shell. It does this by using a special form of the **bsh** command name: `-bsh`. When called with an initial hyphen (`-`), the shell first reads and runs commands found in the system `/etc/profile` file and your `$HOME/.profile`, if one exists. The `/etc/profile` file sets variables needed by all users. Finally, the shell is ready to read commands from your standard input.

If the **File** [*Parameter*] parameter is specified when the Bourne shell is started, the shell runs the script file identified by the **File** parameter, including any parameters specified. The script file specified must have read permission; any `setuid` and `setgid` settings are ignored. The shell then reads the commands. If either the **-c** or **-s** flag is used, do not specify a script.

Related concepts

Available shells
The following are the shells that are provided with AIX.

Bourne shell environment

All variables (with their associated values) known to a command at the beginning of its execution constitute its *environment*. This environment includes variables that a command inherits from its parent process and variables specified as keyword parameters on the command line that calls the command.

The shell passes to its child processes the variables named as arguments to the built-in **export** command. This command places the named variables in the environments of both the shell and all its future child processes.

Keyword parameters are variable-value pairs that appear in the form of assignments, normally before the procedure name on a command line (but see also the flag for the **set** command). These variables are placed in the environment of the procedure being called.

See the following examples:

- Consider the following procedure, which displays the values of two variables (saved in a command file named `key_command`):

  ```
  # key_command
  echo $a $b
  ```

  The following command lines produce the output shown:

  ```
  Input                              Output
  a=key1   b=key2  key_command       key1 key2
  a=tom    b=john  key_command       tom john
  ```

A procedure's keyword parameters are not included in the parameter count stored in $*#*.

A procedure can access the values of any variables in its environment. If it changes any of these values, however, the changes are not reflected in the shell environment. The changes are local to the procedure in question. To place the changes in the environment that the procedure passes to its child processes, you must export the new values within that procedure.

See the following examples:

- To obtain a list of variables that are exportable from the current shell, type the following:

  ```
  export
  ```

- To obtain a list of read-only variables from the current shell, type the following:

```
readonly
```

- To obtain a list of variable-value pairs in the current environment, type the following:

```
env
```

For more information about user environments, see "/etc/environment file" on page 320.

Conditional substitution in the Bourne shell

Normally, the shell replaces the expression $*Variable* with the string value assigned to the *Variable* variable, if there is one. However, there is a special notation that allows *conditional substitution*, depending on whether the variable is set or not null, or both.

By definition, a variable is set if it has ever been assigned a value. The value of a variable can be the null string, which you can assign to a variable in any one of the following ways:

Item	Description
`A=`	
`bcd=""`	
`Efg=''`	Assigns the null string to the `A`, `bcd`, and `Efg`.
`set '' ""`	Sets the first and second positional parameters to the null string and unsets all other positional parameters.

The following is a list of the available expressions you can use to perform conditional substitution:

Item	Description
${*Variable***-** *String***}**	If the variable is set, substitute the *Variable* value in place of this expression. Otherwise, replace this expression with the *String* value.
${*Variable***:-***String***}**	If the variable is set and not null, substitute the *Variable* value in place of this expression. Otherwise, replace this expression with the *String* value.
${*Variable***=***String***}**	If the variable is set, substitute the *Variable* value in place of this expression. Otherwise, set the *Variable* value to the *String* value and then substitute the *Variable* value in place of this expression. You cannot assign values to positional parameters in this fashion.
${*Variable***:=***String***}**	If the variable is set and not null, substitute the *Variable* value in place of this expression. Otherwise, set the *Variable* value to the *String* value and then substitute the *Variable* value in place of this expression. You cannot assign values to positional parameters in this fashion.

${*Variable***?***String***}**	If the variable is set, substitute the *Variable* value in place of this expression. Otherwise, display a message of the following form:
	`Variable: String`
	and exit from the current shell (unless the shell is the login shell). If you do not specify a value for the *String* variable, the shell displays the following message:
	`Variable: parameter null or not set`

Item	Description

${*Variable***:?***String***}**	If the variable is set and not null, substitute the *Variable* value in place of this expression. Otherwise, display a message of the following form:
	`Variable: String`
	and exit from the current shell (unless the shell is the login shell). If you do not specify the *String* value, the shell displays the following message:
	`Variable: parameter null or not set`

${*Variable***+***String***}**	If the variable is set, substitute the *String* value in place of this expression. Otherwise, substitute the null string.
${*Variable***:+***String***}**	If the variable is set and not null, substitute the *String* value in place of this expression. Otherwise, substitute the null string.

In conditional substitution, the shell does not evaluate the *String* variable until the shell uses this variable as a substituted string. Thus, in the following example, the shell executes the **pwd** command only if d is not set or is null:

```
echo ${d:-`pwd`}
```

Related concepts

User-defined variables in the Bourne shell
The Bourne shell recognizes alphanumeric variables to which string values can be assigned.

Positional parameters in the Bourne shell

When you run a shell procedure, the shell implicitly creates positional parameters that reference each word on the command line by its position on the command line.

The word in position 0 (the procedure name) is called $0, the next word (the first parameter) is called $1, and so on, up to $9. To refer to command line parameters numbered higher than 9, use the built-in **shift** command.

You can reset the values of the positional parameters explicitly by using the built-in **set** command.

Note: When an argument for a position is not specified, its positional parameter is set to null. Positional parameters are global and can be passed to nested shell procedures.

Related concepts

User-defined variables in the Bourne shell
The Bourne shell recognizes alphanumeric variables to which string values can be assigned.

Related reference

Predefined special variables in the Bourne shell
Several variables have special meanings. The following variables are set only by the Bourne shell:

File name substitution in the Bourne shell

The Bourne shell permits you to perform file name substitutions.

Command parameters are often file names. You can automatically produce a list of file names as parameters on a command line. To do this, specify a character that the shell recognizes as a pattern-matching character. When a command includes such a character, the shell replaces it with the file names in a directory.

Note: The Bourne shell does not support file name expansion based on equivalence classification of characters.

Most characters in such a pattern match themselves, but you can also use some special pattern-matching characters in your pattern. These special characters are as follows:

Item	Description
*****	Matches any string, including the null string
?	Matches any one character
[. . .]	Matches any one of the characters enclosed in square brackets
[! . . .]	Matches any character within square brackets *other than* one of the characters that follow the exclamation mark

Within square brackets, a pair of characters separated by a hyphen (-) specifies the set of all characters lexicographically within the inclusive range of that pair, according to the binary ordering of character values.

Pattern matching has some restrictions. If the first character of a file name is a dot (.), it can be matched only by a pattern that also begins with a dot. For example, * matches the file names *myfile* and *yourfile* but not the file names *.myfile* and *.yourfile*. To match these files, use a pattern such as the following:

```
.*file
```

If a pattern does not match any file names, then the pattern itself is returned as the result of the attempted match.

File and directory names should not contain the characters *, ?, [, or] because they can cause infinite recursion (that is, infinite loops) during pattern-matching attempts.

Input and output redirection in the Bourne shell

There are redirection options that can be used in commands.

In general, most commands do not know whether their input or output is associated with the keyboard, the display screen, or a file. Thus, a command can be used conveniently either at the keyboard or in a pipeline.

The following redirection options can appear anywhere in a simple command. They can also precede or follow a command, but are not passed to the command.

Item	Description
<*File*	Uses the specified file as standard input.
>*File*	Uses the specified file as standard output. Creates the file if it does not exist; otherwise, truncates it to zero length.
> >*File*	Uses the specified file as standard output. Creates the file if it does not exist; otherwise, adds the output to the end of the file.
<<[-]*eofstr*	Reads as standard input all lines from the *eofstr* variable up to a line containing only *eofstr* or up to an end-of-file character. If any character in the *eofstr* variable is quoted, the shell does not expand or interpret any characters in the input lines. Otherwise, it performs variable and command substitution and ignores a quoted newline character (**\newline**). Use a backslash (\) to quote characters within the *eofstr* variable or within the input lines. If you add a hyphen (-) to the << redirection option, then all leading tabs are stripped from the *eofstr* variable and from the input lines.
<&*Digit*	Associates standard input with the file descriptor specified by the *Digit* variable.
>&*Digit*	Associates standard output with the file descriptor specified by the *Digit* variable.
<&-	Closes standard input.

Item	Description
>&-	Closes standard output.

Note: The restricted shell does not allow output redirection.

For more information about redirection, see "Input and output redirection" on page 348.

List of Bourne shell built-in commands

The following is a list of Bourne shell built-in commands.

Item	Description
:	Returns a zero exit value
.	Reads and executes commands from a file parameter and then returns.
break	Exits from the enclosing **for**, **while**, or **until** command loops, if any.
cd	Changes the current directory to the specified directory.
continue	Resumes the next iteration of the enclosing **for**, **while**, or **until** command loops.
echo	Writes character strings to standard output.
eval	Reads the arguments as input to the shell and executes the resulting command or commands.
exec	Executes the command specified by the **Argument** parameter, instead of this shell, without creating a new process.
exit	Exits the shell whose exit status is specified by the **n** parameter.
export	Marks names for automatic export to the environment of subsequently executed commands.
hash	Finds and remembers the location in the search path of specified commands.
pwd	Displays the current directory.
read	Reads one line from standard input.
readonly	Marks name specified by **Name** parameter as read-only.
return	Causes a function to exit with a specified return value.
set	Controls the display of various parameters to standard output.
shift	Shifts command-line arguments to the left.
test	Evaluates conditional expressions.
times	Displays the accumulated user and system times for processes run from the shell.
trap	Runs a specified command when the shell receives a specified signal or signals.
type	Interprets how the shell would interpret a specified name as a command name.
ulimit	Displays or adjusts allocated shell resources.
umask	Determines file permissions.
unset	Removes the variable or function corresponding to a specified name.
wait	Waits for the specified child process to end and reports its termination status.

Related reference

Bourne shell built-in commands

Special commands are built into the Bourne shell and run in the shell process.

Bourne shell commands

You can issue commands in the Bourne shell.

When you issue a command in the Bourne shell, it first evaluates the command and makes all indicated substitutions. It then runs the command provided that:

- The command name is a Bourne shell special built-in command.

  OR

- The command name matches the name of a defined function. If this is the case, the shell sets the positional parameters to the parameters of the function.

If the command name matches neither a built-in command nor the name of a defined function and the command names an executable file that is a compiled (binary) program, the shell (as *parent*) creates a new (*child*) process that immediately runs the program. If the file is marked executable but is not a compiled program, the shell assumes that it is a shell procedure. In this case, the shell creates another instance of itself (a *subshell*), to read the file and execute the commands included in it. The shell also runs a parenthesized command in a subshell. To the user, a compiled program is run in exactly the same way as a shell procedure. The shell normally searches for commands in file system directories in this order:

1. `/usr/bin`
2. `/etc`
3. `/usr/sbin`
4. `/usr/ucb`
5. `$HOME/bin`
6. `/usr/bin/X11`
7. `/sbin`
8. Current directory

The shell searches each directory, in turn, continuing with the next directory if it fails to find the command.

Note: The *PATH* variable determines the order in which the shell searches directories. You can change the particular sequence of directories searched by resetting the *PATH* variable.

If you give a specific path name when you run a command (for example, `/usr/bin/sort`), the shell does not search any directories other than the one you specify. If the command name contains a slash (**/**), the shell does not use the search path.

You can give a full path name that begins with the root directory (such as `/usr/bin/sort`). You can also specify a path name relative to the current directory. If you specify, for example:

```
bin/myfile
```

the shell looks in the current directory for a directory named `bin` and in that directory for the file `myfile`.

Note: The restricted shell does not run commands containing a slash (/).

The shell remembers the location in the search path of each executed command (to avoid unnecessary **exec** commands later). If it finds the command in a relative directory (one whose name does not begin with **/**), the shell must redetermine the command's location whenever the current directory changes. The shell forgets all remembered locations each time you change the *PATH* variable or run the **hash -r** command.

Character quotation
Many characters have a special meaning to the shell. Sometimes you want to conceal that meaning. Single (') and double (") quotation marks surrounding a string, or a backslash (\) before a single character allow you to conceal the character's meaning.

All characters (except the enclosing single quotation marks) are taken literally, with any special meaning removed. Thus, the command:

```
stuff='echo $? $*; ls * | wc'
```

assigns the literal string `echo $? $*; ls * | wc` to the variable `stuff`. The shell does not execute the **echo**, **ls**, and **wc** commands or expand the $? and $* variables and the asterisk (*) special character.

Within double quotation marks, the special meaning of the dollar sign ($), backquote (`), and double quotation (") characters remains in effect, while all other characters are taken literally. Thus, within double quotation marks, command and variable substitution takes place. In addition, the quotation marks do not affect the commands within a command substitution that is part of the quoted string, so characters there retain their special meanings.

Consider the following sequence:

```
ls *
file1 file2 file3
message="This directory contains `ls * ` "
echo $message
This directory contains file1 file2 file3
```

This shows that the asterisk (*) special character inside the command substitution was expanded.

To hide the special meaning of the dollar sign ($), backquote (`), and double quotation (") characters within double quotation marks, precede these characters with a backslash (\). When you do not use double quotation marks, preceding a character with a backslash is equivalent to placing it within single quotation marks. Therefore, a backslash immediately preceding a newline character (that is, a backslash at the end of the line) hides the newline character and allows you to continue the command line on the next physical line.

Signal handling
The shell ignores **INTERRUPT** and **QUIT** signals for an invoked command if the command is terminated with an ampersand (&); that is, if it is running in the background. Otherwise, signals have the values inherited by the shell from its parent, with the exception of the **SEGMENTATION VIOLATION** signal.

For more information, see the Bourne shell built-in **trap** command.

Bourne shell compound commands
A compound command is one of the following.

- Pipeline (one or more simple commands separated by the pipe (|) symbol)
- List of simple commands
- Command beginning with a reserved word
- Command beginning with the control operator left parenthesis (()

Unless otherwise stated, the value returned by a compound command is that of the last simple command executed.

Reserved words
The following reserved words for the Bourne shell are recognized only when they appear without quotation marks as the first word of a command.

```
for           do             done
case          esac
if            then            fi
elif          else
while         until
```

```
{               }
(               )
```

Item	Description
for*Identifier* [**in***Word*. . .] **do***List* **done**	Sets the *Identifier* parameter to the word or words specified by the *Word* parameter (one at a time) and runs the commands specified in the *List* parameter. If you omit **in** *Word*. . ., then the **for** command runs the *List* parameter for each positional parameter that is set, and processing ends when all positional parameters have been used.
case *Word* **in** *Pattern* [| *Pattern*] . . . **)** *List***;;** [*Pattern* [|*Pattern*] . . . **)** *List***;;**] **... esac**	Runs the commands specified in the *List* parameter that are associated with the first *Pattern* parameter that matches the value of the *Word* parameter. Uses the same character-matching notation in patterns that are used for file name substitution, except that a slash (/), leading dot (.), or a dot immediately following a slash (/.) do not need to match explicitly.
if *List* **then** *List* [**elif** *List* **then** *List*] . . . [**else** *List*] **fi**	Runs the commands specified in the *List* parameter following the **if** command. If the command returns a zero exit value, the shell runs the *List* parameter following the first **then** command. Otherwise, it runs the *List* parameter following the **elif** command (if it exists). If this exit value is zero, the shell runs the List parameter following the next **then** command. If the command returns a nonzero exit value, the shell runs the *List* parameter following the **else** command (if it exists). If no **else** *List* or **then** *List* is performed, the **if** command returns a zero exit value.
while *List* **do** *List* **done**	Runs the commands specified in the *List* parameter following the **while** command. If the exit value of the last command in the **while** *List* is zero, the shell runs the *List* parameter following the **do** command. It continues looping through the lists until the exit value of the last command in the **while** *List* is nonzero. If no commands in the **do** *List* are performed, the **while** command returns a zero exit value.
until *List* **do** *List* **done**	Runs the commands specified in the *List* parameter following the **until** command. If the exit value of the last command in the **until** *List* is nonzero, runs the *List* following the **do** command. Continues looping through the lists until the exit value of the last command in the **until** *List* is zero. If no commands in the **do** *List* are performed, the **until** command returns a zero exit value.
(*List* **)**	Runs the commands in the *List* parameter in a subshell.
{ *List***; }**	Runs the commands in the *List* parameter in the current shell process and does not start a subshell.
Name **() {** *List* **}**	Defines a function that is referenced by the *Name* parameter. The body of the function is the list of commands between the braces specified by the *List* parameter.

Bourne shell built-in commands
Special commands are built into the Bourne shell and run in the shell process.

Unless otherwise indicated, output is written to file descriptor 1 (standard output) and the exit status is 0 (zero) if the command does not contain any syntax errors. Input and output redirection is permitted.

The following special commands are treated somewhat differently from other special built-in commands:

```
: (colon)     exec          shift
. (dot)       exit          times
break         export        trap
continue      readonly      wait
eval          return
```

The Bourne shell processes these commands as follows:

- Keyword parameter assignment lists preceding the command remain in effect when the command completes.
- I/O redirections are processed after parameter assignments.
- Errors in a shell script cause the script to stop processing.

Related reference

List of Bourne shell built-in commands

The following is a list of Bourne shell built-in commands.

Special command descriptions

The Bourne shell provides the following special built-in commands.

Item	Description
:	Returns a zero exit value.
. *File*	Reads and runs commands from the *File* parameter and returns. Does not start a subshell. The shell uses the search path specified by the *PATH* variable to find the directory containing the specified file.
break [*n*]	Exits from the enclosing **for**, **while**, or **until** command loops, if any. If you specify the *n* variable, the **break** command breaks the number of levels specified by the *n* variable.
continue [*n*]	Resumes the next iteration of the enclosing **for**, **while**, or **until** command loops. If you specify the *n* variable, the command resumes at the n^{th} enclosing loop.
cd *Directory*]	Changes the current directory to *Directory*. If you do not specify *Directory*, the value of the *HOME* shell variable is used. The *CDPATH* shell variable defines the search path for *Directory*. *CDPATH* is a colon-separated list of alternative directory names. A null path name specifies the current directory (which is the default path). This null path name appears immediately after the equal sign in the assignment or between the colon delimiters anywhere else in the path list. If *Directory* begins with a slash (/), the shell does not use the search path. Otherwise, the shell searches each directory in the *CDPATH* shell variable. **Note:** The restricted shell cannot run the **cd** shell command.
echo *String* . . .]	Writes character strings to standard output. See the **echo** command for usage and parameter information. The **-n** flag is not supported.
eval [*Argument* . . .]	Reads arguments as input to the shell and runs the resulting command or commands.
exec [*Argument* . . .]	Runs the command specified by the *Argument* parameter in place of this shell without creating a new process. Input and output arguments can appear, and if no other arguments appear, cause the shell input or output to be modified. This is not recommended for your login shell.
exit [*n*]	Causes a shell to exit with the exit value specified by the *n* parameter. If you omit this parameter, the exit value is that of the last command executed (the Ctrl-D key sequence also causes a shell to exit). The value of the *n* parameter can be from 0 to 255, inclusive.
export [*Name* . . .]	Marks the specified names for automatic export to the environments of subsequently executed commands. If you do not specify the *Name* parameter, the **export** command displays a list of all names that are exported in this shell. You cannot export function names.

Item	Description
hash [**-r**] [*Command* . . .]	Finds and remembers the location in the search path of each *Command* specified. The **-r** flag causes the shell to forget all locations. If you do not specify the flag or any commands, the shell displays information about the remembered commands in the following format: `Hits  Cost  Command` `Hits` indicates the number of times a command has been run by the shell process. `Cost` is a measure of the work required to locate a command in the search path. `Command` shows the path names of each specified command. Certain situations require that the stored location of a command be recalculated; for example, the location of a relative path name when the current directory changes. Commands for which that might be done are indicated by an asterisk (`*`) next to the `Hits` information. `Cost` is incremented when the recalculation is done.
pwd	Displays the current directory. See the **pwd** command for a discussion of command options.
read [*Name* . . .]	Reads one line from standard input. Assigns the first word in the line to the first *Name* parameter, the second word to the second *Name* parameter, and so on, with leftover words assigned to the last *Name* parameter. This command returns a value of 0 unless it encounters an end-of-file character.
readonly [*Name* . . .]	Marks the name specified by the *Name* parameter as read-only. The value of the name cannot be reset. If you do not specify any *Name*, the **readonly** command displays a list of all read-only names.
return [*n*]	Causes a function to exit with a return value of *n*. If you do not specify the *n* variable, the function returns the status of the last command performed in that function. This command is valid only when run within a shell function.

Item	Description
set [*Flag* [*Argument*] . . .]	Sets one or more of the following flags: **-a** Marks for export all variables to which an assignment is performed. If the assignment precedes a command name, the export attribute is effective only for that command execution environment, except when the assignment precedes one of the special built-in commands. In this case, the export attribute persists after the built-in command has completed. If the assignment does not precede a command name, or if the assignment is a result of the operation of the **getopts** or **read** commands, the export attribute persists until the variable is unset. **-e** Exits immediately if all of the following conditions exist for a command: • It exits with a return value greater than 0 (zero). • It is not part of the compound list of a **while**, **until**, or **if** command. • It is not being tested using AND or OR lists. • It is not a pipeline preceded by the ! (exclamation point) reserved word. **-f** Disables file name substitution. **-h** Locates and remembers the commands called within functions as the functions are defined. (Normally, these commands are located when the function is performed; see the **hash** command.) **-k** Places all keyword parameters in the environment for a command, not just those preceding the command name. **-n** Reads commands but does not run them. To check for shell script syntax errors, use the **-n** flag. **-t** Exits after reading and executing one command. **-u** Treats an unset variable as an error and immediately exits when performing variable substitution. An interactive shell does not exit. **-v** Displays shell input lines as they are read. **-x** Displays commands and their arguments before they are run. **—** Does not change any of the flags. This is useful in setting the **$1** positional parameter to a string beginning with a hyphen (-). Using a plus sign (+) rather than a hyphen (-) unsets flags. You can also specify these flags on the shell command line. The *$-* special variable contains the current set of flags. Any *Argument* to the **set** command becomes a positional parameter and is assigned, in order, to $1, $2, **. . .**, and so on. If you do not specify a *flag* or *Argument*, the **set** command displays all the names and values of the current shell variables.

Item	Description
shift [*n*]	Shifts command line arguments to the left; that is, reassigns the value of the positional parameters by discarding the current value of **$1** and assigning the value of **$2** to **$1**, of **$3** to **$2**, and so on. If there are more than 9 command line arguments, the 10th is assigned to **$9** and any that remain are still unassigned (until after another **shift**). If there are 9 or fewer arguments, the **shift** command unsets the highest-numbered positional parameter that has a value. The **$0** positional parameter is never shifted. The **shift** *n* command is a shorthand notation specifying *n* number of consecutive shifts. The default value of the *n* parameter is 1.
test *Expression* | [*Expression*]	Evaluates conditional expressions. See the **test** command for a discussion of command flags and parameters. The **-h** flag is not supported by the built-in test command in **bsh**.
times	Displays the accumulated user and system times for processes run from the shell.
trap [*Command*] [*n*] . . .	Runs the command specified by the *Command* parameter when the shell receives the signal or signals specified by the *n* parameter. The **trap** commands are run in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. **Note:** The shell scans the *Command* parameter once when the trap is set and again when the trap is taken. If you do not specify a command, then all traps specified by the *n* parameter are reset to their current values. If you specify a null string, this signal is ignored by the shell and by the commands it invokes. If the *n* parameter is zero (0), the specified command is run when you exit from the shell. If you do not specify either a command or a signal, the **trap** command displays a list of commands associated with each signal number.
type [*Name* . . .]	Indicates how the shell would interpret it as a command name for each *Name* specified.

Item	Description
ulimit [**-HS**] [**-c** | **-d** | **-f** | **-m** | **-r** | **-s** | **-t** |**-u**] [*limit*]	Displays or adjusts allocated shell resources. The shell resource settings can be displayed either individually or as a group. The default mode is to display resources set to the soft setting, or the lower bound, as a group. The setting of shell resources depends on the effective user ID of the current shell. The hard level of a resource can be set only if the effective user ID of the current shell is root. You will get an error if you are not root user and you are attempting to set the hard level of a resource. By default, the root user sets both the hard and soft limits of a particular resource. The root user should therefore be careful in using the **-S**, **-H**, or default flag usage of limit settings. Unless you are a root user, you can set only the soft limit of a resource. After a limit has been decreased by a nonroot user, it cannot be increased, even back to the original system limit. To set a resource limit, select the appropriate flag and the limit value of the new resource, which should be an integer. You can set only one resource limit at a time. If more than one resource flag is specified, you receive undefined results. By default, **ulimit** with only a new value on the command line sets the file size of the shell. Use of the **-f** flag is optional. You can specify the following **ulimit** command flags: **-c** Sets or displays core segment for shell. **-d** Sets or displays data segment for shell. **-f** Sets or displays file size for shell. **-H** Sets or displays hard resource limit (root user only). **-m** Sets or displays memory for shell. **-r** Sets or displays maximum number of threads per process. **-s** Sets or displays stack segment for shell. **-S** Sets or displays soft resource limit. **-t** Sets or displays CPU time maximum for shell. **-u** Sets or displays maximum number of processes per user.
umask [*nnn*]	Determines file permissions. This value, along with the permissions of the creating process, determines a file's permissions when the file is created. The default is 022. When no value is entered, **umask** displays the current value.
unset [*Name . . .*]	Removes the corresponding variable or function for each name specified by the *Name* parameter. The *PATH*, *PS1*, *PS2*, *MAILCHECK*, and *IFS* shell variables cannot be unset.
wait [*n*]	Waits for the child process whose process number is specified by the *n* parameter to exit and then returns the exit status of that process. If you do not specify the *n* parameter, the shell waits for all currently active child processes, and the return value is 0.

Command substitution in the Bourne shell
Command substitution allows you to capture the output of any command as an argument to another command.

When you place a command line within backquotes (` `), the shell first runs the command or commands and then replaces the entire expression, including the backquotes, with the output. This feature is often used to give values to shell variables. For example, the statement:

```
today=`date`
```

assigns the string representing the current date to the *today* variable. The following assignment saves, in the *files* variable, the number of files in the current directory:

```
files=`ls | wc -l`
```

You can perform command substitution on any command that writes to standard output.

To nest command substitutions, precede each of the nested backquotes with a backslash (\), as in:

```
logmsg=`echo Your login directory is \`pwd\``
```

You can also give values to shell variables indirectly by using the **read** special command. This command takes a line from standard input (usually your keyboard) and assigns consecutive words on that line to any variables named. For example:

```
read first init last
```

takes an input line of the form:

```
J. Q. Public
```

and has the same effect as if you had typed:

```
first=J. init=Q. last=Public
```

The **read** special command assigns any excess words to the last variable.

Variable substitution in the Bourne shell
The Bourne shell permits you to perform variable substitutions.

The Bourne shell has several mechanisms for creating variables (assigning a string value to a name). Certain variables, positional parameters and keyword parameters are normally set only on a command line. Other variables are simply names to which you or the shell can assign string values.

Related concepts
Unattended terminals
All systems are vulnerable if terminals are left logged in and unattended. The most serious problem occurs when a system manager leaves a terminal unattended that has been enabled with root authority. In general, users should log out anytime they leave their terminals.

User-defined variables in the Bourne shell
The Bourne shell recognizes alphanumeric variables to which string values can be assigned.

To assign a string value to a name, type the following:

```
Name=String
```

A name is a sequence of letters, digits, and underscores that begins with an underscore or a letter. To use the value that you have assigned to a variable, add a dollar sign ($) to the beginning of its name. Thus, the *$Name* variable yields the value specified by the *String* variable. Note that no spaces are on either side of the equal sign (=) in an assignment statement. (Positional parameters cannot appear in an assignment statement. You can put more than one assignment on a command line, but remember that the shell performs the assignments from right to left.

If you enclose the *String* variable with double or single quotation marks (" or '), the shell does not treat blanks, tabs, semicolons, and newline characters within the string as word delimiters, but it imbeds them literally in the string.

If you enclose the *String* variable with double quotation marks ("), the shell still recognizes variable names in the string and performs variable substitution; that is, it replaces references to positional parameters and other variable names that are prefaced by dollar sign ($) with their corresponding values, if any. The shell also performs command substitution within strings that are enclosed in double quotation marks.

If you enclose the *String* variable with single quotation marks ('), the shell does not substitute variables or commands within the string. The following sequence illustrates this difference:

```
You:            num=875
                number1="Add $num"
                number2='Add $num'
                echo $number1
System:         Add 875
You:            echo $number2
System:         Add $num
```

The shell does not reinterpret blanks in assignments after variable substitution. Thus, the following assignments result in $first and $second having the same value:

```
first='a string with embedded blanks'
second=$first
```

When you reference a variable, you can enclose the variable name (or the digit designating a positional parameter) in braces { } to delimit the variable name from any string following. In particular, if the character immediately following the name is a letter, digit, or underscore, and the variable is not a positional parameter, then the braces are required:

```
You:            a='This is a'
                echo "${a}n example"
System:         This is an example
You:            echo "$a test"
System:         This is a test
```

Related concepts

Positional parameters in the Bourne shell
When you run a shell procedure, the shell implicitly creates positional parameters that reference each word on the command line by its position on the command line.

Related reference

Conditional substitution in the Bourne shell
Normally, the shell replaces the expression $*Variable* with the string value assigned to the *Variable* variable, if there is one. However, there is a special notation that allows *conditional substitution*, depending on whether the variable is set or not null, or both.

Variables used by the Bourne shell
The shell uses the following variables. Although the shell sets some of them, you can set or reset all of them.

Item	Description
CDPATH	Specifies the search path for the **cd** (change directory) command.
HOME	Indicates the name of your *login directory*, which is the directory that becomes the current directory upon completion of a login. The login program initializes this variable. The **cd** command uses the value of the *$HOME* variable as its default value. Using this variable rather than an explicit path name in a shell procedure allows the procedure to be run from a different directory without alterations.

Item	Description
IFS	The characters that are IFS (internal field separators), which are the characters that the shell uses during blank interpretation. The shell initially sets the *IFS* variable to include the blank, tab, and newline characters.
LANG	Determines the locale to use for the locale categories when both the *LC_ALL* variable and the corresponding environment variable (beginning with `LC_`) do not specify a locale.
LC_ALL	Determines the locale to be used to override any values for locale categories specified by the settings of the *LANG* environment variable or any environment variables beginning with `LC_`.
LC_COLLATE	Defines the collating sequence to use when sorting names and when character ranges occur in patterns.
LC_CTYPE	Determines the locale for the interpretation of sequences of bytes of text data as characters (that is, single versus multibyte characters in arguments and input files), which characters are defined as letters (**alpha** character class), and the behavior of character classes within pattern matching.
LC_MESSAGES	Determines the language in which messages should be written.
LIBPATH	Specifies the search path for shared libraries.
LOGNAME	Specifies your login name, marked `readonly` in the `/etc/profile` file.
MAIL	Indicates the path name of the file used by the mail system to detect the arrival of new mail. If this variable is set, the shell periodically checks the modification time of this file and displays the value of `$MAILMSG` if the time changes and the length of the file is greater than 0. Set the *MAIL* variable in the `.profile` file. The value normally assigned to it by users of the **mail** command is `/usr/spool/mail/$LOGNAME`.
MAILCHECK	The number of seconds that the shell lets elapse before checking again for the arrival of mail in the files specified by the *MAILPATH* or *MAIL* variables. The default value is 600 seconds (10 minutes). If you set the *MAILCHECK* variable to 0, the shell checks before each prompt.
MAILMSG	The mail notification message. If you explicitly set the *MAILMSG* variable to a null string (`MAILMSG=" "`), no message is displayed.
MAILPATH	A list of file names separated by colons. If this variable is set, the shell informs you of the arrival of mail in any of the files specified in the list. You can follow each file name by a % and a message to be displayed when mail arrives. Otherwise, the shell uses the value of the *MAILMSG* variable or, by default, the message `[YOU HAVE NEW MAIL]`. **Note:** When the *MAILPATH* variable is set, these files are checked instead of the file set by the *MAIL* variable. To check the files set by the *MAILPATH* variable and the file set by the *MAIL* variable, specify the `MAIL` file in your list of `MAILPATH` files.

Item	Description
PATH	The search path for commands, which is an ordered list of directory path names separated by colons. The shell searches these directories in the specified order when it looks for commands. A null string anywhere in the list represents the current directory. The *PATH* variable is normally initialized in the `/etc/environment` file, usually to `/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:/sbin`. You can reset this variable to suit your own needs. The *PATH* variable provided in your `.profile` file also includes `$HOME/bin` and your current directory. If you have a project-specific directory of commands, for example, `/project/bin`, that you want searched before the standard system directories, set your *PATH* variable as follows: `PATH=/project/bin:$PATH` The best place to set your *PATH* variable to a value other than the default value is in your `$HOME/.profile` file. You cannot reset the *PATH* variable if you are executing commands under the restricted shell.
PS1	The string to be used as the primary system prompt. An interactive shell displays this prompt string when it expects input. The default value of the *PS1* variable is $ followed by a blank space for nonroot users.
PS2	The value of the secondary prompt string. If the shell expects more input when it encounters a newline character in its input, it prompts with the value of the *PS2* variable. The default value of the *PS2* variable is > followed by a blank space.
SHACCT	The name of a file that you own. If this variable is set, the shell writes an accounting record in the file for each shell script executed. You can use accounting programs such as `acctcom` and `acctcms` to analyze the data collected.
SHELL	The path name of the shell, which is kept in the environment. This variable should be set and exported by the `$HOME/.profile` file of each restricted login.
TIMEOUT	The number of minutes a shell remains inactive before it exits. If this variable is set to a value greater than zero (0), the shell exits if a command is not entered within the prescribed number of seconds after issuing the PS1 prompt. (Note that the shell can be compiled with a maximum boundary that cannot be exceeded for this value.) A value of zero indicates no time limit.

Related concepts

Blank interpretation

After the shell performs variable and command substitution, it scans the results for internal field separators (those defined in the *IFS* shell variable).

Predefined special variables in the Bourne shell

Several variables have special meanings. The following variables are set only by the Bourne shell:

Item	Description
$@	Expands the positional parameters, beginning with **$1**. Each parameter is separated by a space. If you place double quotation marks (" ") around **$@**, the shell considers each positional parameter a separate string. If no positional parameters exist, the Bourne shell expands the statement to an unquoted null string.

Item	Description
$*	Expands the positional parameters, beginning with **$1**. The shell separates each parameter with the first character of the *IFS* variable value. If you place double quotation marks (" ") around **$***, the shell includes the positional parameter values, in double quotation marks. Each value is separated by the first character of the *IFS* variable.
$#	Specifies the number of positional parameters passed to the shell, not counting the name of the shell procedure itself. The *$#* variable thus yields the number of the highest-numbered positional parameter that is set. One of the primary uses of this variable is to check for the presence of the required number of arguments. Only positional parameters **$0** through **$9** are accessible through the shell.
$?	Specifies the exit value of the last command executed. Its value is a decimal string. Most commands return a value of 0 to indicate successful completion. The shell itself returns the current value of the *$?* variable as its exit value.

$$	Identifies the process number of the current process. Because process numbers are unique among all existing processes, this string is often used to generate unique names for temporary files.
	
	The following example illustrates the recommended practice of creating temporary files in a directory used only for that purpose:
	
	`temp=/tmp/$$`
	`ls >$temp`
	`.`
	`.`
	`.`
	`rm $temp`

$!	Specifies the process number of the last process run in the background using the & terminator.
$-	A string consisting of the names of the execution flags currently set in the shell.

Related concepts

Positional parameters in the Bourne shell
When you run a shell procedure, the shell implicitly creates positional parameters that reference each word on the command line by its position on the command line.

Blank interpretation
After the shell performs variable and command substitution, it scans the results for internal field separators (those defined in the *IFS* shell variable).

The shell splits the line into distinct words at each place it finds one or more of these characters separating each distinct word with a single space. It then retains explicit null arguments (`""` or `''`) and discards implicit null arguments (those resulting from parameters that have no values).

Related reference

Variables used by the Bourne shell
The shell uses the following variables. Although the shell sets some of them, you can set or reset all of them.

C shell

The C shell is an interactive command interpreter and a command programming language. It uses syntax that is similar to the C programming language.

The **csh** command starts the C shell.

When you log in, the **csh** command first searches the system-wide setup file `/etc/csh.cshrc`. If the setup file exists, the C shell executes the commands stored in that file. Next, the C shell executes the system-wide setup file `/etc/csh.login` if it is available. Then, it searches your home directory for

the `.cshrc` and `.login` files. If they exist, they contain any customized user information pertinent to running the C shell. All variables set in the `/etc/csh.cshrc` and `/etc/csh.login` files might be overridden by your `.cshrc` and `.login` files in your `$HOME` directory. Only the root user can modify the `/etc/csh.cshrc` and `/etc/csh.login` files.

The `/etc/csh.login` and `$HOME/.login` files are executed only once at login time. These files are generally used to hold environment variable definitions, commands that you want executed once at login, or commands that set up terminal characteristics.

The `/etc/csh.cshrc` and `$HOME/.cshrc` files are executed at login time and every time the **csh** command or a C shell script is invoked. They are generally used to define C shell characteristics, such as aliases and C shell variables (for example, *history*, *noclobber*, or *ignoreeof*). It is recommended that you only use the C shell built-in commands in the `/etc/csh.cshrc` and `$HOME/.cshrc` files because using other commands increases the startup time for shell scripts.

Related reference

C shell built-in commands list
The following are C shell built-in commands.

C shell limitations

The following are limitations of the C shell.

- Words can be no longer than 1024 bytes.
- Argument lists are limited to *ARG_MAX* bytes. Values for the *ARG_MAX* variable are found in the `/usr/include/sys/limits.h` file.
- The number of arguments to a command that involves file name expansion is limited to 1/6th the number of bytes allowed in an argument list.
- Command substitutions can substitute no more bytes than are allowed in an argument list.
- To detect looping, the shell restricts the number of alias substitutions on a single line to 20.
- The **csh** command does not support file name expansion based on equivalence classification of characters.
- File descriptors (other than standard in, standard out, and standard error) opened before **csh** executes any application are not available to that application.

Alias substitution in the C shell

An *alias* is a name assigned to a command or command string. The C shell allows you to assign aliases and use them as you would commands. The shell maintains a list of the aliases that you define.

After the shell scans the command line, it divides the commands into distinct words and checks the first word of each command, left to right, to see if there is an alias. If an alias is found, the shell uses the history mechanism to replace the text of the alias with the text of the command referenced by the alias. The resulting words replace the command and argument list. If no reference is made to the history list, the argument list is left unchanged.

The **alias** and **unalias** built-in commands establish, display, and modify the alias list. Use the alias command in the following format:

```
alias [Name [WordList]]
```

The optional *Name* variable specifies the alias for the specified name. If you specify a word list with the *WordList* variable, the command assigns it as the alias of the *Name* variable. If you run the **alias** command without either optional variable, it displays all C shell aliases.

If the alias for the **ls** command is **ls -l**, the following command:

```
ls /usr
```

is replaced by the command:

```
ls -l /usr
```

The argument list is undisturbed because there is no reference to the history list in the command with an alias. Similarly, if the alias for the **lookup** command is as follows:

```
grep \!^ /etc/passwd
```

then the shell replaces `lookup bill` with the following:

```
grep bill /etc/passwd
```

In this example, `!^` refers to the history list, and the shell replaces it with the first argument in the input line, in this case `bill`.

You can use special pattern-matching characters in an alias. The following command:

```
alias lprint 'pr &bslash2.!* >
> print'
```

creates a command that formats its arguments to the line printer. The `!` character is protected from the shell in the alias by use of single quotation marks so that the alias is not expanded until the **pr** command runs.

If the shell locates an alias, it performs the word transformation of the input text and begins the alias process again on the reformed input line. If the first word of the next text is the same as the previous text, then looping is prevented by flagging the alias to terminate the alias process. Other subsequent loops are detected and result in an error.

Related concepts

History substitution in the C shell

History substitution lets you modify individual words from previous commands to create new commands. History substitution makes it easy to repeat commands, repeat the arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing.

Variable substitution in the C shell

The C shell maintains a set of variables, each of which has as its value a list of zero or more words. Some of these variables are set by the shell or referred to by it. For example, the *argv* variable is an image of the shell variable list, and words that comprise the value of this variable are referred to in special ways.

To change and display the values of variables, use the **set** and **unset** commands. Of the variables referred to by the shell, a number are toggles (variables that turn something on and off). The shell does not examine toggles for a value, only for whether they are set or unset. For example, the *verbose* shell variable is a toggle that causes command input to be echoed. The setting of this variable results from issuing the **-v** flag on the command line.

Other operations treat variables numerically. The **@** command performs numeric calculations, and the result is assigned to a variable. Variable values are, however, always represented as (zero or more) strings. For numeric operations, the null string is considered to be zero, and the second and subsequent words of multi-word values are ignored.

When you issue a command, the shell parses the input line and performs alias substitution. Next, before running the command, it performs variable substitution. The $ character keys the substitution. It is, however, passed unchanged if followed by a blank, tab, or newline character. Preceding the $ character with a \ prevents this expansion, except in two cases:

- The command is enclosed in `" "`. In this case, the shell always performs the substitution.
- The command is enclosed in `' '`. In this case, the shell never performs the substitution. Strings enclosed in `' '` are interpreted for command substitution.

The shell recognizes input and output redirection before variable expansion and expands each separately. Otherwise, the command name and complete argument list expand together. It is therefore possible for the first (command) word to generate more than one word, the first of which becomes the command name, and the rest of which become parameters.

Unless enclosed in " " or given the :q modifier, the results of variable substitution might eventually be subject to command and file name substitution. When enclosed by double quotation marks, a variable with a value that consists of multiple words expands to a single word or a portion of a single word, with the words of the variable's value separated by blanks. When you apply the :q modifier to a substitution, the variable expands to multiple words. Each word is separated by a blank and enclosed in double quotation marks to prevent later command or file name substitution.

The following notations allow you to introduce variable values into the shell input. Except as noted, it is an error to reference a variable that is not set with the **set** command.

You can apply the modifiers :gh, :gt, :gr, :h, :r, :q, and :x to the following substitutions. If { } appear in the command form, then the modifiers must be placed within the braces. Only one **:** modifier is permitted on each variable expansion.

Item	Description
$*Name*	
${*Name***}**	Replaced by the words assigned to the Name variable, each separated by a blank. Braces insulate the *Name* variable from any following characters that would otherwise be part of it. Shell variable names start with a letter and consist of up to 20 letters and digits, including the underline (_) character. If the Name variable does not specify a shell variable but is set in the environment, then its value is returned. The modifiers preceded by colons, as well as the other forms described here, are not available in this case.
$*Name***[***number***]**	
${*Name***[***number***]}**	Selects only some of the words from the value of the *Name* variable. The number is subjected to variable substitution and might consist of a single number or two numbers separated by a hyphen (-). The first word of a variable's string value is numbered 1. If the first number of a range is omitted, it defaults to 1. If the last number of a range is omitted, it defaults to **$#***Name*. The asterisk (*) symbol selects all words. It is not an error for a range to be empty if the second argument is omitted or is in a range.
$#*Name*	
${#*Name***}**	Gives the number of words in the *Name* variable. This can be used in a **[***number***]** as shown above. For example, $*Name*[$#*Name*].
$0	Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.
$*number*	
${*number***}**	Equivalent to $argv[*number*].
$*	Equivalent to $argv[*].

The following substitutions may not be changed with **:** modifiers:

Item	Description
$?*name*	
${?*name***}**	Substitutes the string 1 if the *name* variable is set, zero (0) if this variable is not set.
$?0	Substitutes 1 if the current input file name is known, zero (0) if the file name is not known.
$$	Substitutes the (decimal) process number of the parent shell.
$<	Substitutes a line from standard input, without further interpretation. Use this substitution to read from the keyboard in a shell procedure.

Related concepts

Command substitution in the C shell

In *command substitution*, the shell executes a specified command and replaces that command with its output.

File name substitution in the C shell

The C Shell permits you to do file name substitutions.

The C shell provides several shortcuts to save time and keystrokes. If a word contains any of the characters `*`, `?`, `[ ]`, or `{ }`, or begins with a tilde (~), that word is a candidate for file name substitution. The C shell regards the word as a pattern and replaces the word with an alphabetized list of file names matching the pattern.

The current collating sequence is used, as specified by the *LC_COLLATE* or *LANG* environment variables. In a list of words specifying file name substitution, an error results if no patterns match an existing file name. However, it is not required that every pattern match. Only the character-matching symbols `*`, `?`, and `[ ]` indicate pattern-matching or file name expansion. The tilde (~) and `{ }` characters indicate file name abbreviation.

File name expansion in the C shell

The asterisk (*) character matches any string of characters, including the null string.

For example, in a directory containing the files:

```
a aa aax alice b bb c cc
```

the command **echo a*** prints all files names beginning with the character a:

```
a aa aax alice
```

Note: When file names are matched, the characters dot (`.`) and slash (/) must be matched explicitly.

The question mark (?) character matches any single character. The following command:

```
ls a?x
```

lists every file name beginning with the letter a, followed by a single character, and ending with the letter x:

```
aax
```

To match a single character or a range of characters, enclose the character or characters inside of `[ ]`. The following command:

```
ls [abc]
```

lists all file names exactly matching one of the enclosed characters:

```
a b c
```

Within brackets, a lexical range of characters is indicated by `[a-z]`. The characters matching this pattern are defined by the current collating sequence.

File name abbreviation in the C shell

The tilde (~) and { characters indicate file name abbreviation. A ~ at the beginning of a file name is used to represent home directories. Standing alone, the ~ character expands to your home directory as reflected in the value of the *home* shell variable.

For example, the following command:

```
ls ~
```

lists all files and directories located in your $HOME directory.

When the command is followed by a name consisting of letters, digits, and hyphen (-) characters, the shell searches for a user with that name and substitutes that user's $HOME directory.

Note: If the ~ character is followed by a character other than a letter or slash (/), or appears anywhere except at the beginning of a word, it does not expand.

To match characters in file names without typing the entire file name, use { } around the file names. The pattern a{b,c,d}e is another way of writing abe ace ade. The shell preserves the left-to-right order and separately stores the results of matches at a low level to preserve this order. This construct might be nested. Thus, the following:

```
~source/s1/{oldls,ls}.c
```

expands to:

```
/usr/source/s1/oldls.c /usr/source/s1/ls.c
```

if the home directory for **source** is /usr/source. Similarly, the following:

```
../{memo,*box}
```

might expand to:

```
../memo ../box ../mbox
```

Note: memo is not sorted with the results of matching *box. As a special case, the {, }, and { } characters are passed undisturbed.

Character classes in the C shell

You can use character classes to match file names within a range indication.

The following format instructs the system to match any single character belonging to the specified class:

```
[:charclass:]
```

The following classes correspond to ctype subroutines:

Character Class	Definition
alnum	Alphanumeric characters
alpha	Uppercase and lowercase letters
cntrl	Control characters
digit	Digits
graph	Graphic characters
lower	Lowercase letters
print	Printable characters
punct	Punctuation character
space	Space, horizontal tab, carriage return, newline, vertical tab, or form-feed character
upper	Uppercase characters
xdigit	Hexadecimal digits

Suppose that you are in a directory containing the following files:

```
a aa aax Alice b bb c cc
```

Type the following command at a C shell prompt:

```
ls [:lower:]
```

The C shell lists all file names that begin with lowercase characters:

```
a aa aax b bb c cc
```

For more information about character class expressions, see the **ed** command.

Environment variables in the C shell

Certain variables have special meaning to the C shell. Of these, *argv*, *cwd*, *home*, *path*, *prompt*, *shell*, and *status* are always set by the shell.

Except for the *cwd* and *status* variables, the action of being set by the shell occurs only at initialization. All of the above variables maintain their settings unless you explicitly reset them.

The **csh** command copies the *USER*, *TERM*, *HOME*, and *PATH* environment variables into the *csh* variables, *user*, *term*, *home*, and *path*, respectively. The values are copied back into the environment whenever the normal shell variables are reset. The *path* variable cannot be set in other than in the **.cshrc** file because **csh** subprocesses import the path definition from the environment and reexport it if changed.

The following variables have special meanings:

Item	Description
argv	Contains the arguments passed to shell scripts. Positional parameters are substituted from this variable.
cdpath	Specifies a list of alternate directories to be searched by the **chdir** or **cd** command to find subdirectories.
cwd	Specifies the full path name of the current directory.
echo	Set when the **-x** command line flag is used; when set, causes each command and its arguments to echo just before being run. For commands that are not built-in, all expansions occur before echoing. Built-in commands are echoed before command and file name substitution because these substitutions are then done selectively.
histchars	Specifies a string value to change the characters used in history substitution. Use the first character of its value as the history substitution character, this replaces the default character, `!`. The second character of its value replaces the ^ character in quick substitutions. **Note:** Setting the **histchars** value to a character used in command or file names might cause unintentional history substitution.
history	Contains a numeric value to control the size of the history list. Any command that is referenced within the number of events permitted is not discarded. Very large values of the *history* variable might cause the shell to run out of memory. Regardless of whether this variable is set, the C shell always saves the last command that ran on the history list.
home	Indicates your home directory initialized from the environment. The file name expansion of the tilde (~) character refers to this variable.
ignoreeof	Specifies that the shell ignore an end-of-file character from input devices that are workstations. This prevents shells from accidentally being killed when the shell reads an end-of-file character (Ctrl-D).
mail	Specifies the files where the shell checks for mail. This is done after each command completion which results in a prompt if a specified time interval has elapsed. The shell displays the message `Mail in file` if the file exists with an access time less than its change time.

Item	Description
	If the first word of the value of the *mail* variable is numeric, it specifies a different mail-checking time interval (in seconds); the default is 600 (10 minutes). If you specify multiple mail files, the shell displays the message `New mail in file`, when there is mail in the specified file.
noclobber	Places restrictions on output redirection to ensure that files are not accidentally destroyed and that redirections append to existing files.
noglob	Inhibits file name expansion. This is most useful in shell scripts that do not deal with file names or when a list of file names has been obtained and further expansions are not desirable.
nonomatch	Specifies that no error results if a file name expansion does not match any existing files; rather, the primitive pattern returns. It is still an error for the primitive pattern to be malformed.
notify	Specifies that the shell send asynchronous notification of changes in job status. The default presents status changes just before displaying the shell prompt.
path	Specifies directories in which commands are sought for execution. A null word specifies the current directory. If there is no *path* variable set, then only full path names can run. The default search path (from the `/etc/environment` file used during login) is as follows: `/usr/bin /etc /usr/sbin /usr/ucb /usr/bin/X11 /sbin` A shell given neither the **-c** nor the **-t** flag normally hashes the contents of the directories in the *path* variable after reading the `.cshrc` and each time the *path* variable is reset. If new commands are added to these directories while the shell is active, you must give the **rehash** command. Otherwise, the commands might not be found.
prompt	Specifies the string displayed before each command is read from an interactive workstation input. If a `!` appears in the string, it is replaced by the current event number. If the `!` character is in a quoted string enclosed by single or double quotation marks, the `!` character must be preceded by a `\`. The default prompt for users without root authority is `%` . The default prompt for the user with root authority is *#*.
savehist	Specifies a numeric value to control the number of entries of the history list that are saved in the `~/.history` file when you log out. Any command referenced in this number of events is saved. During startup, the shell reads `~/.history` into the history list, enabling history to be saved across logins. Very large values of the *savehist* variable slow down the shell startup.
shell	Specifies the file in which the C shell resides. This is used in forking shells to interpret files that have execute bits set, but which are not executable by the system. This is initialized to the home of the C shell.
status	Specifies the status returned by the last command. If the command ends abnormally, `0200` is added to the status. Built-in commands that are unsuccessful return an exit status of 1. Successful built-in commands set status to a value of 0.
time	Controls automatic timing of commands. If this variable is set, any command that takes more than the specified number of CPU seconds will display a line of resources used at the end of execution. For more information about the default outputs, see the built-in **time** command.
verbose	Set by the **-v** command line flag, this variable causes the words of each command to display after history substitution.

Job control in the C shell

The shell associates a job number with each process. The shell keeps a table of current jobs and assigns them small integer numbers.

When you start a job in the background with an ampersand (&) , the shell prints a line that looks like the following:

```
[1] 1234
```

This line indicates that the job number is 1 and that the job is composed of a single process with a process ID of 1234. Use the built-in **jobs** command to see the table of current jobs.

A job running in the background competes for input if it tries to read from the workstation. Background jobs can also produce output for the workstation that gets interleaved with the output of other jobs.

You can refer to jobs in the shell in several ways. Use the percent (%) character to introduce a job name. This name can be either the job number or the command name that started the job, if this name is unique. For example, if a **make** process is running as job 1, you can refer to it as %1. You can also refer to it as %make if there is only one suspended job with a name that begins with the string make. You can also use the following:

```
%?String
```

to specify a job whose name contains the *String* variable, if there is only one such job.

The shell detects immediately whenever a process changes its state. If a job becomes blocked so that further progress is impossible, the shell sends a message to the workstation. This message displays only after you press the Enter key. If, however, the *notify* shell variable is set, the shell immediately issues a message that indicates changes in the status of background jobs. Use the built-in **notify** command to mark a single process so that its status changes are promptly reported. By default, the **notify** command marks the current process.

C shell built-in commands list

The following are C shell built-in commands.

Item	Description
@	Displays the value of specified shell variables.
alias	Displays specified aliases or all aliases.
bg	Puts the current or specified jobs into the background.
break	Resumes running after the end of the nearest enclosing **foreach** or **while** command.
breaksw	Breaks from a **switch** command.
case	Defines a label in a **switch** command.
cd	Changes the current directory to the specified directory.
chdir	Changes the current directory to the specified directory.
continue	Continues execution of the nearest enclosing **foreach** or **while** command.
default	Labels the default case in a **switch** statement.
dirs	Displays the directory stack.
echo	Writes character strings to the standard output of the shell.
else	Runs the commands that follow the second `else` in an `if (`*Expression*`) then` ...`else if (`*Expression2*`) then` ... `else` ... `endif` command sequence.
end	Signifies the end of a sequence of commands preceded by the **foreach** command.
endif	Runs the commands that follow the second `then` statement in an `if (`*Expression*`) then` ... `else if (`*Expression2*`) then` ... `else` ... `endif` command sequence.

Item	Description
endsw	Marks the end of a **switch** (*String*) `case` *String* `:` ... `breaksw default:` ... `breaksw` `endsw` command sequence. This command sequence successively matches each case label against the value of the *String* variable. Execution continues after the **endsw** command if a **breaksw** command is executed or if no label matches and there is no default.
eval	Reads variable values as input to the shell and executes the resulting command or commands in the context of the current shell.
exec	Runs the specified command in place of the current shell.
exit	Exits the shell with either the value of the status shell variable or the value of the specified expression.
fg	Brings the current or specified jobs into the foreground, continuing them if they are stopped.
foreach	Successively sets a *Name* variable for each member specified by the *List* variable and a sequence of commands, until reaching an **end** command.
glob	Displays list using history, variable, and file name expansion.
goto	Continues to run after a specified line.
hashstat	Displays statistics indicating how successful the hash table has been at locating commands.
history	Displays the history event list.
if	Runs a specified command if a specified expression is true.
jobs	Lists the active jobs.
kill	Sends either the **TERM** (terminate) signal or the signal specified by the *Signal* variable to the specified job or process.
limit	Limits usage of a specified resource by the current process and each process it creates.
login	Ends a login shell and replaces it with an instance of the `/usr/sbin/login` command.
logout	Ends a login shell.
nice	Sets the priority of commands run in the shell.
nohup	Causes hangups to be ignored for the remainder of a procedure.
notify	Causes the shell to notify you asynchronously when the status of the current or a specified job changes.
onintr	Controls the action of the shell on interrupts.
popd	Pops the directory stack and returns to the new top directory.
pushd	Exchanges elements of the directory stack.
rehash	Causes recomputation of the internal hash table containing the contents of the directories in the path shell variable.
repeat	Runs the specified command, subject to the same restrictions as the **if** command, the number of times specified.
set	Shows the value of all shell variables.
setenv	Modifies the value of the specified environment variable.
shift	Shifts the specified variable to the left.
source	Reads command specified by the *Name* variable.
stop	Stops the current or specified jobs running in the background.

Item	Description
suspend	Stops the shell as if a **STOP** signal has been received.
switch	Starts a **switch** (*String*) `case` *String* `:` ... `breaksw default:` ... `breaksw endsw` command sequence. This command sequence successively matches each case label against the value of the *String* variable. If none of the labels match before a default label is found, the execution begins after the default label.
time	Displays a summary of the time used by the shell and its child processes.
umask	Determines file permissions.
unalias	Discards all aliases with names that match the *Pattern* variable.
unhash	Disables the use of the internal hash table to locate running programs.
unlimit	Removes resource limitations.
unset	Removes all variables having names that match the *Pattern* variable.
unsetenv	Removes all variables from the environment whose names match the specified *Pattern* variable.
wait	Waits for all background jobs.
while	Evaluates the commands between the `while` and the matching `end` command sequence while an expression specified by the *Expression* variable evaluates nonzero.

The following is related information:

Korn shell

The **ksh** and **stty** commands.

The **alias**, **cd**, **export**, **fc**, **getopts**, **read**, **set**, and **typeset** Korn shell commands.

The `/etc/passwd` file.

Bourne shell

The **bsh** or **Rsh** command, **login** command.

The Bourne shell **read** special command.

The `setuid` subroutine, `setgid` subroutine.

The `null` special file.

The `environment` file, `profile` file format.

C shell

The **csh** command, **ed** command.

The **alias**, **unalias**, **jobs**, **notify** and **set** C Shell built-in commands.

Related concepts

C shell

The C shell is an interactive command interpreter and a command programming language. It uses syntax that is similar to the C programming language.

C shell built-in commands

Built-in commands are run within the shell. If a built-in command occurs as any component of a pipeline, except the last, the command runs in a subshell.

Signal handling in the C shell

The C shell normally ignores quit signals. Jobs running detached are not affected by signals generated from the keyboard (**INTERRUPT**, **QUIT**, and **HANGUP**).

Other signals have the values the shell inherits from its parent. You can control the shell's handling of **INTERRUPT** and **TERMINATE** signals in shell procedures with **onintr**. Login shells catch or ignore **TERMINATE** signals depending on how they are set up. Shells other than login shells pass **TERMINATE** signals on to the child processes. In no cases are **INTERRUPT** signals allowed when a login shell is reading the `.logout` file.

C shell commands

A simple command is a sequence of words separated by blanks or tabs. A *word* is a sequence of characters or numerals, or both, that does not contain blanks without quotation marks.

In addition, the following characters and doubled characters also form single words when used as command separators or terminators:

```
&       |      ;
&&      ||     <<        > >
<       >      (          )
```

These special characters can be parts of other words. Preceding them with a backslash (\), however, prevents the shell from interpreting them as special characters. Strings enclosed in ' ' or " " (matched pairs of quotation characters) or backquotes can also form parts of words. Blanks, tab characters, and special characters do not form separate words when they are enclosed in these marks. In addition, you can enclose a newline character within these marks by preceding it with a backslash (\).

The first word in the simple command sequence (numbered 0) usually specifies the name of a command. Any remaining words, with a few exceptions, are passed to that command. If the command specifies an executable file that is a compiled program, the shell immediately runs that program. If the file is marked executable but is not a compiled program, the shell assumes that it is a shell script. In this case, the shell starts another instance of itself (a subshell) to read the file and execute the commands included in it.

C shell built-in commands

Built-in commands are run within the shell. If a built-in command occurs as any component of a pipeline, except the last, the command runs in a subshell.

Note: If you enter a command from the C shell prompt, the system searches for a built-in command first. If a built-in command does not exist, the system searches the directories specified by the *path* shell variable for a system-level command. Some C shell built-in commands and operating system commands have the same name. However, these commands do not necessarily work the same way. For more information on how the command works, check the appropriate command description.

If you run a shell script from the shell, and the first line of the shell script begins with `#!` / *ShellPathname*, the C shell runs the shell specified in the comment to process the script. Otherwise, it runs the default shell (the shell linked to `/usr/bin/sh`). If run by the default shell, C shell built-in commands might not be recognized. To run C shell commands, make the first line of the script `#!/usr/bin/csh`.

Related reference

C shell built-in commands list

The following are C shell built-in commands.

C shell command descriptions
The C shell provides the following built-in commands.

Item	Description
alias [*Name* [*WordList*]]	Displays all aliases if you do not specify any parameters. Otherwise, the command displays the alias for the specified *Name*. If *WordList* is specified, this command assigns the value of *WordList* to the alias *Name*. The specified alias *Name* cannot be `alias` or `unalias`.
bg [%*Job* ...]	Puts the current job or job specified by *Job* into the background, continuing the job if it was stopped.
break	Resumes running after the `end` of the nearest enclosing **foreach** or **while** command.
breaksw	Breaks from a **switch** command; resumes after the **endsw** command.
case *Label*:	Defines a *Label* in a **switch** command.
cd[*Name*]	Equivalent to the **chdir** command (see following description).
chdir [*Name*]	Changes the current directory to that specified by the *Name* variable. If you do not specify *Name*, the command changes to your home directory. If the value of the *Name* variable is not a subdirectory of the current directory and does not begin with `/`, `./`, or `../`, the shell checks each component of the ***cdpath*** shell variable to see if it has a subdirectory matching the *Name* variable. If the *Name* variable is a shell variable with a value that begins with a slash (/), the shell tries this to see if it is a directory. The **chdir** command is equivalent to the **cd** command.
continue	Continues execution at the `end` of the nearest enclosing **while** or **foreach** command.
default:	Labels the `default` case in a `switch` statement. The **default** should come after all other `case` labels.
dirs	Displays the directory stack.
echo	Writes character strings to the standard output of the shell.
else	Runs the commands that follow the second `else` in an `if` (*Expression*) `then` ...`else if` (*Expression2*) `then` ... `else` ... `endif` command sequence. **Note:** The **else** statement is the **csh** built-in command when using the **if(***expr***) then** ..**else** ...**endif**. If the **(***expr***)** is true, then the commands up to the **else** statement is executed. If the **(***expr***)** is false, then the commands between the **else** and **endif** statement are executed. Anything in single quotes is taken literally and not interpreted.

Item	Description
end	Successively sets the *Name* variable to each member specified by the *List* variable and runs the sequence of *Commands* between the **foreach** and the matching **end** statements. The **foreach** and **end** statements must appear alone on separate lines. Uses the `continue` statement to continue the loop and the `break` statement to end the loop prematurely. When the **foreach** command is read from the terminal, the C shell prompts with a ? to allow *Commands* to be entered. Commands within loops, prompted for by ?, are not placed in the history list.
endif	If the *Expression* variable is true, runs the *Commands* that follow the first `then` statement. If the `else if` *Expression2* is true, runs the *Commands* that follow the second `then` statement. If the `else if` *Expression2* is false, runs the *Commands* that follow the `else`. Any number of `else if` pairs are possible. Only one `endif` statement is needed. The `else` segment is optional. The words `else` and `endif` can be used only at the beginning of input lines. The `if` segment must appear alone on its input line or after an `else` command.
endsw	Successively matches each **case** label against the value of the *string* variable. The *string* is command and file name expanded first. Use the pattern-matching characters *, ?, and `[ . . . ]` in the **case** labels, which are variable-expanded. If none of the labels match before a **default** label is found, the execution begins after the **default** label. The **case** label and the **default** label must appear at the beginning of the line. The **breaksw** command causes execution to continue after the **endsw** command. Otherwise, control might fall through the **case** and **default** labels, as in the C programming language. If no label matches and there is no **default**, execution continues after the **endsw** command.
eval *Parameter . . .*	Reads the value of the *Parameter* variable as input to the shell and runs the resulting command or commands in the context of the current shell. Use this command to run commands generated as the result of command or variable substitution because parsing occurs before these substitutions.
exec *Command*	Runs the specified *Command* in place of the current shell.
exit (*Expression***)**	Exits the shell with either the value of the ***status*** shell variable (if no *Expression* is specified) or with the value of the specified *Expression*.
fg [%*Job* ...]	Brings the current job or job specified by *Job* into the foreground, continuing the job if it was stopped.
foreach *Name* (*List*) *Command. . .*	Successively sets a *Name* variable for each member specified by the *List* variable and a sequence of commands, until reaching an **end** command.
glob *List*	Displays *List* using history, variable, and file name expansion. Puts a null character between words and does not include a carriage return at the end.
goto *Word*	Continues to run after the line specified by the *Word* variable. The specified *Word* is file name and command expanded to yield a string of the form specified by the *Label:* variable. The shell rewinds its input as much as possible and searches for a line of the form *Label***:**, possibly preceded by blanks or tabs.

Item	Description
hashstat	Displays statistics indicating how successful the hash table has been at locating commands.
history [**-r** | **-h**] [*n*]	Displays the history event list. The oldest events are displayed first. If you specify a number *n*, only the specified number of the most recent events are displayed. The **-r** flag reverses the order in which the events are displayed so the most recent is displayed first. The **-h** flag displays the history list without leading numbers. Use this flag to produce files suitable for use with the **-h** flag of the **source** command.
if (*Expression*) *Command*	Runs the specified *Command* (including its arguments) if the specified *Expression* is true. Variable substitution on the *Command* variable happens early, at the same time as the rest of the `if` statement. The specified *Command* must be a simple command (rather than a pipeline, command list, or parenthesized command list). **Note:** Input and output redirection occurs even if the *Expression* variable is false and the *Command* is not executed.
jobs [**-l**]	Lists the active jobs. With the **-l** (lowercase *L*) flag, the **jobs** command lists process IDs in addition to the job number and name.
kill -l | [[-*Signal*] % *Job*...| *PID*...]	Sends either the **TERM** (terminate) signal or the signal specified by *Signal* to the specified *Job* or *PID* (process). Specify signals either by number or by name (as given in the `/usr/include/sys/signal.h` file, stripped of the **SIG** prefix). The **-l** (lowercase *L*) flag lists the signal names.

Item	Description
limit [**-h**] [*Resource* [*Max-Use*]]	Limits the usage of the specified resource by the current process and each process it creates. Process resource limits are defined in the `/etc/security/limits` file. Controllable resources are the central processing unit (CPU) time, file size, data size, core dump size, and memory use. Maximum allowable values for these resources are set with the **mkuser** command when the user is added to the system. They are changed with the **chuser** command. Limits are categorized as either soft or hard. Users may increase their soft limits up to the ceiling imposed by the hard limits. You must have root user authority to increase a soft limit above the hard limit, or to change hard limits. The **-h** flag displays hard limits instead of the soft limits. If a *Max-Use* parameter is not specified, the **limit** command displays the current limit of the specified resource. If the *Resource* parameter is not specified, the **limit** command displays the current limits of all resources. For more information about the resources controlled by the **limit** subcommand, see the `getrlimit, setrlimit, or vlimit` subroutine. The *Max-Use* parameter for CPU time is specified in the *hh*:*mm*:*ss* format. The *Max-Use* parameter for other resources is specified as a floating-point number or an integer optionally followed by a scale factor. The scale factor is k or kilobytes (1024 bytes), m or megabytes, or b or blocks (the units used by the `ulimit` subroutine. If you do not specify a scale factor, k is assumed for all resources. For both resource names and scale factors, unambiguous prefixes of the names suffice. **Note:** This command limits the physical memory (memory use) available for a process only if there is contention for system memory by other active processes.
login	Ends a login shell and replaces it with an instance of the `/usr/bin/login` command. This is one way to log out (included for compatibility with the **ksh** and **bsh** commands).
logout	Ends a login shell. This command must be used if the `ignoreeof` option is set.
nice [**+***n*] [*Command*]	If no values are specified, sets the priority of commands run in this shell to 24. If the ***+n*** flag is specified, sets the priority plus the specified number. If the ***+n*** flag and *Command* are specified, runs *Command* at priority 24 plus the specified number. If you have root user authority, you can run the **nice** statement with a negative number. The *Command* always runs in a subshell, and the restrictions placed on commands in simple `if` statements apply.
nohup [*Command*]	Causes **hangups** to be ignored for the remainder of the script when no *Command* is specified. If *Command* is specified, causes the specified *Command* to be run with **hangups** ignored. To run a pipeline or list of commands, put the pipeline or list in a shell script, give the script execute permission, and use the shell script as the value of the *Command* variable. All processes run in the background with an ampersand (&) are effectively protected from being sent a **hangup** signal when you log out. However, these processes are still subject to explicitly sent **hangups** unless the **nohup** statement is used.

Item	Description
notify [**%***Job*...]	Causes the shell to notify you asynchronously when the status of the current job or specified *Job* changes. Normally, the shell provides notification just before it presents the shell prompt. This feature is automatic if the ***notify*** shell variable is set.
onintr [**-** | *Label*]	Controls the action of the shell on interrupts. If no arguments are specified, restores the default action of the shell on interrupts, which ends shell scripts or returns to the command input level. If a **-** flag is specified, causes all interrupts to be ignored. If *Label* is specified, causes the shell to run a `goto` *Label* statement when the shell receives an interrupt or when a child process ends due to an interruption. In any case, if the shell is running detached and interrupts are being ignored, all forms of the **onintr** statement have no meaning. Interrupts continue to be ignored by the shell and all invoked commands.
popd [**+***n*]	Pops the directory stack and changes to the new top directory. If you specify a ***+n*** variable, the command discards the n^{th} entry in the stack. The elements of the directory stack are numbered from the top, starting at 0.
pushd [**+***n*|*Name*]	With no arguments, exchanges the top two elements of the directory stack. With the *Name* variable, the command changes to the new directory and pushes the old current directory (as given in the ***cwd*** shell variable) onto the directory stack. If you specify a ***+n*** variable, the command rotates the n^{th} component of the directory stack around to be the top element and changes to it. The members of the directory stack are numbered from the top, starting at 0.
rehash	Causes recomputation of the internal hash table of the contents of the directories in the ***path*** shell variable. This action is needed if new commands are added to directories in the ***path*** shell variable while you are logged in. The **rehash** command is necessary only if commands are added to one of the user's own directories or if someone changes the contents of one of the system directories.
repeat *Count Command*	Runs the specified *Command*, subject to the same restrictions as commands in simple `if` statements, the number of times specified by *Count*. **Note:** I/O redirections occur exactly once, even if the *Count* variable equals 0.
set [[*Name*[*n*]] [**=** *Word*]] | [*Name* = (*List*)]	Shows the value of all shell variables when used with no arguments. Variables that have more than a single word as their value are displayed as a parenthesized word list. If only *Name* is specified, the C shell sets the *Name* variable to the null string. Otherwise, sets *Name* to the value of the *Word* variable, or sets the *Name* variable to the list of words specified by the *List* variable. When *n* is specified, the n^{th} component of the *Name* variable is set to the value of the *Word* variable; the n^{th} component must already exist. In all cases, the value is command and file name expanded. These arguments may be repeated to set multiple values in a single **set** command. However, variable expansion happens for all arguments before any setting occurs.

Item	Description
setenv*Name Value*	Sets the value of the environment variable specified by the *Name* variable to *Value*, a single string. The most commonly used environment variables, ***USER***, ***TERM***, ***HOME***, and ***PATH***, are automatically imported to and exported from the C shell variables ***user***, ***term***, ***home***, and ***path***. There is no need to use the **setenv** statement for these.
shift [*Variable*]	Shifts the members of the ***argv*** shell variable or the specified *Variable* to the left. An error occurs if the **argv** shell variable or specified *Variable* is not set or has less than one word as its value.
source[**-h**] *Name*	Reads commands written in the *Name* file. You can nest the **source** commands. However, if they are nested too deeply, the shell might run out of file descriptors. An error in a **source** command at any level ends all nested **source** commands. Normally, input during **source** commands is not placed on the history list. The **-h** flag causes the commands to be placed in the history list without executing them.
stop [**%***Job* ...]	Stops the current job or specified *Job* running in the background.
suspend	Stops the shell as if a **STOP** signal had been received.
switch (*string*)	Starts a **switch** (*String*) **case** *String* **:** ... **breaksw default:** ... **breaksw endsw** command sequence. This command sequence successively matches each case label against the value of the *String* variable. If none of the labels match before a default label is found, the execution begins after the default label.

Item	Description

time [*Command*]	The **time** command controls automatic timing of commands. If you do not specify the *Command* variable, the **time** command displays a summary of time used by this shell and its children. If you specify a command with the *Command* variable, it is timed. The shell then displays a time summary, as described under the ***time*** shell variable. If necessary, an extra shell is created to display the time statistic when the command completes.
	
	The following example uses **time** with the **sleep** command:
	
	`time sleep`
	
	The output from this command looks similar to the following:
	
	`0.0u 0.0s 0:00 100% 44+4k 0+0io 0pf+0w`
	
	The output fields are as follows:
	
	First
	Number of seconds of CPU time devoted to the user process
	
	Second
	Number of seconds of CPU time consumed by the kernel on behalf of the user process
	
	Third
	Elapsed (wall clock) time for the command
	
	Fourth
	Total user CPU Time plus system time, as a percentage of elapsed time
	
	Fifth
	Average amount of shared memory used, plus average amount of unshared data space used, in kilobytes
	
	Sixth
	Number of block input and output operations
	
	Seventh
	Page faults plus number of swaps

umask [*Value*]	Determines file permissions. This *Value*, along with the permissions of the creating process, determines a file's permissions when the file is created. The default is 022. The current setting will be displayed if no *Value* is specified.
unalias *|*_Pattern_	Discards all aliases with names that match the *Pattern* variable. All aliases are removed by the **unalias *** command. The absence of aliases does not cause an error.
unhash	Disables the use of the internal hash table to locate running programs.
unlimit [**-h**][*Resource*]	Removes the limitation on the *Resource* variable. If no *Resource* variable is specified, all resource limitations are removed. See the description of the **limit** command for the list of *Resource* names. The **-h** flag removes corresponding hard limits. Only a user with root user authority can change hard limits.

Item	Description
unset *|*Pattern*	Removes all variables with names that match the *Pattern* variable. Use **unset *** to remove all variables. If no variables are set, it does not cause an error.
unsetenv*Pattern*	Removes all variables from the environment whose name matches the specified *Pattern*. (See the **setenv** built-in command.)
wait	Waits for all background jobs. If the shell is interactive, an **INTERRUPT** (usually the Ctrl-C key sequence) disrupts the wait. The shell then displays the names and job numbers of all jobs known to be outstanding.
while (*Expression*) *Command*. . . end	Evaluates the *Commands* between the **while** and the matching **end** statements while the expression specified by the *Expression* variable evaluates nonzero. You can use the **break** statement to end and the **continue** statement to continue the loop prematurely. The **while** and **end** statements must appear alone on their input lines. If the input is from a terminal, prompts occur after the **while** (*Expression*) similar to the **foreach** statement.
@ [*Name*[*n*] = *Expression*]	Displays the values of all the shell variables when used with no arguments. Otherwise, sets the name specified by the *Name* variable to the value of the *Expression* variable. If the expression contains <, >, &, or | characters, this part of the expression must be placed within parentheses. When *n* is specified, the n^{th} component of the *Name* variable is set to the *Expression* variable. Both the *Name* variable and its n^{th} component must already exist. C language operators, such as *= and +=, are available. The space separating the *Name* variable from the assignment operator is optional. Spaces are, however, required in separating components of the *Expression* variable, which would otherwise be read as a single word. Special suffix operators, double plus sign (++) and double hyphen (- -) increase and decrease, respectively, the value of the *Name* variable.

C shell expressions and operators

The **@** built-in command and the **exit**, **if**, and **while** statements accept expressions that include operators similar to those of C language, with the same precedence.

The following operators are available:

Operator	What it means
( )	change precedence
~	complement
!	negation
*/ %	multiply, divide, modulo
+ -	add, subtract
<< > >	left shift, right shift
<= >= < >	relational operators
== != =~ !~	string comparison/pattern matching
&	bitwise AND
^	bitwise exclusive OR

Operator	What it means
|	bitwise inclusive OR
&&	logical AND
||	logical OR

In the previous list, precedence of the operators decreases down the list (left to right, top to bottom).

Note: The operators + and - are right-associative. For example, evaluation of `a + b - c` is performed as follows:

```
a + (b - c)
```

and not as follows:

```
(a + b) - c
```

The `==`, `!=`, `=~`, and `!~` operators compare their arguments as strings; all others operate on numbers. The `=~` and `!~` operators are similar to `==` and `!=`, except that the rightmost side is a *pattern* against which the leftmost operand is matched. This reduces the need for use of the **switch** statement in shell procedures.

The logical operators **or** (||) and **and** (&&) are also available. They can be used to check for a range of numbers, as in the following example:

```
if ($#argv > 2 && $#argv < 7) then
```

In the preceding example, the number of arguments must be greater than 2 and less than 7.

Strings beginning with zero (0) are considered octal numbers. Null or missing arguments are considered 0. All expressions result in strings representing decimal numbers. Note that two components of an expression can appear in the same word. Except when next to components of expressions that are syntactically significant to the parser (`& | < > ( )`), expression components should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in parentheses `( )` and file inquiries of the form (**-operator** *Filename*), where **operator** is one of the following:

Item	Description
r	Read access
w	Write access
x	Execute access
e	Existence
o	Ownership
z	Zero size
f	Plain file
d	Directory

The specified *Filename* is command and file name expanded and then tested to see if it has the specified relationship to the real user. If *Filename* does not exist or is inaccessible, all inquiries return `false(0)`. If the command runs successfully, the inquiry returns a value of `true(1)`. Otherwise, if the command fails, the inquiry returns a value of `false(0)`. If more detailed status information is required, run the command outside an expression and then examine the *status* shell variable.

Command substitution in the C shell
In *command substitution*, the shell executes a specified command and replaces that command with its output.

To perform command substitution in the C shell, enclose the command or command string in backquotes (` `). The shell normally breaks the output from the command into separate words at blanks, tabs, and newline characters. It then replaces the original command with this output.

In the following example, the backquotes (` `) around the **date** command indicate that the output of the command will be substituted:

```
echo The current date and time is: `date`
```

The output from this command might look like the following:

```
The current date and time is: Wed Apr 8 13:52:14 CDT 1992
```

The C shell performs command substitution selectively on the arguments of built-in shell commands. This means that it does not expand those parts of expressions that are not evaluated. For commands that are not built-in, the shell substitutes the command name separately from the argument list. The substitution occurs in a child of the main shell, but only after the shell performs input or output redirection.

If a command string is surrounded by " ", the shell treats only newline characters as word separators, thus preserving blanks and tabs within the word. In all cases, the single final newline character does not force a new word.

Related concepts
Variable substitution in the C shell
The C shell maintains a set of variables, each of which has as its value a list of zero or more words. Some of these variables are set by the shell or referred to by it. For example, the *argv* variable is an image of the shell variable list, and words that comprise the value of this variable are referred to in special ways.

Nonbuilt-in C shell command execution
When the C shell determines that a command is not a built-in shell command, it attempts to run the command with the `execv` subroutine.

Each word in the *path* shell variable names a directory from which the shell attempts to run the command. If given neither the **-c** nor **-t** flag, the shell hashes the names in these directories into an internal table. The shell tries to call the `execv` subroutine on a directory only if there is a possibility that the command resides there. If you turn off this mechanism with the **unhash** command or give the shell the **-c** or **-t** flag, the shell concatenates with the given command name to form a path name of a file. The shell also does this in any case for each directory component of the *path* variable that does not begin with a slash (/). The shell then attempts to run the command.

Parenthesized commands always run in a subshell. For example:

```
(cd ; pwd) ; pwd
```

displays the home directory without changing the current directory location. However, the command:

```
cd ; pwd
```

changes the current directory location to the home directory. Parenthesized commands are most often used to prevent the **chdir** command from affecting the current shell.

If the file has execute permission, but is not an executable binary to the system, then the shell assumes it is a file containing shell commands and runs a new shell to read it.

If there is an alias for the shell, then the words of the alias are prefixed to the argument list to form the shell command. The first word of the alias should be the full path name of the shell.

History substitution in the C shell

History substitution lets you modify individual words from previous commands to create new commands. History substitution makes it easy to repeat commands, repeat the arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing.

History substitutions begin with the exclamation mark (`!`) character and can appear anywhere on the command line, provided they do not nest (in other words, a history substitution cannot contain another history substitution). You can precede the `!` with a `\` to cancel the exclamation point's special meaning. In addition, if you place the `!` before a blank, tab, newline character, =, or `(`, history substitution does not occur.

History substitutions also occur when you begin an input line with a carat (^). The shell echoes any input line containing history substitutions at the workstation before it executes that line.

Related concepts

Alias substitution in the C shell

An *alias* is a name assigned to a command or command string. The C shell allows you to assign aliases and use them as you would commands. The shell maintains a list of the aliases that you define.

History lists for the C shell

The history list saves commands that the shell reads from the command line that consist of one or more words. History substitution reintroduces sequences of words from these saved commands into the input stream.

The *history* shell variable controls the size of the history list. You must set the *history* shell variable either in the `.cshrc` file or on the command line with the built-in **`set`** command. The previous command is always retained regardless of the value of the *history* variable. Commands in the history list are numbered sequentially, beginning with 1. The built-in **`history`** command produces output similar to the following:

```
9 write michael
10 ed write.c
11 cat oldwrite.c
12 diff *write.c
```

The shell displays the command strings with their event numbers. The event number appears to the left of the command and represent when the command was entered in relation to the other commands in the history. It is not usually necessary to use event numbers to refer to events, but you can have the current event number displayed as part of your system prompt by placing an exclamation mark (`!`) in the prompt string assigned to the *PROMPT* environment variable.

A full history reference contains an event specification, a word designator, and one or more modifiers in the following general format:

```
Event[.]Word:Modifier[:Modifier] . . .
```

Note: Only one word can be modified. A string that contains blanks is not allowed.

In the previous sample of **`history`** command output, the current event number is 13. Using this example, the following refer to previous events:

Item	Description
`!10`	Event number 10.
`!-2`	Event number 11 (the current event minus 2).
`!d`	Command word beginning with d (event number 12).
`!?mic?`	Command word containing the string `mic` (event number 9).

These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case, `!!` refers to the previous command; the command **`!!`** alone on an input line reruns the previous command.

Event specification for the C shell
To select words from an event, follow the event specification with a colon (`:`) and one of the following word designators (the words of an input line are numbered sequentially starting from 0)

Item	Description
0	First word (the command name)
n	n^{th} argument
^	First argument
$	Last argument
%	Word matched by an immediately preceding ?*string*? search
x-y	Range of words from the x^{th} word to the y^{th} word
-y	Range of words from the first word (0) to the y^{th} word
*****	First through the last argument, or nothing if there is only one word (the command name) in the event
x*	x^{th} argument through the last argument
***x*-**	Same as *x** but omitting the last argument

If the word designator begins with a ^, $, *, -, or %, you can omit the colon that separates the event specification from the word designator. You can also place a sequence of the following modifiers after the optional word designator, each preceded by a colon:

Item	Description
h	Removes a trailing path name extension, leaving the head.
r	Removes a trailing *.xxx* component, leaving the root name.
e	Removes all but the *.xxx* trailing extension.
s/*OldWord***/***NewWord***/**	Substitutes the value of the *NewWord* variable for the value of the *OldWord* variable.

The left side of a substitution is not a pattern in the sense of a string recognized by an editor; rather, it is a word, a single unit without blanks. Normally, a slash (/) delimits the original word (*OldWord*) and its replacement (*NewWord*). However, you can use any character as the delimiter. In the following example, using the % as a delimiter allows a / to be included in the words:

```
s%/home/myfile%/home/yourfile%
```

The shell replaces an ampersand (&) with the *OldWord* text in the *NewWord* variable. In the following example, `/home/myfile` becomes `/temp/home/myfile`.

```
s%/home/myfile%/temp&%
```

The shell replaces a null word in a substitution with either the last substitution or with the last string used in the contextual scan `!?`*String*`?`. You can omit the trailing delimiter (/) if a newline character follows immediately. Use the following modifiers to delimit the history list:

Item	Description
t	Removes all leading path name components, leaving the tail
&	Repeats the previous substitution
g	Applies the change globally; that is, all occurrences for each line
p	Displays the new command, but does not run it

Item	Description
q	Quotes the substituted words, thus preventing further substitutions
x	Acts like the **q** modifier, but breaks into words at blanks, tabs, and newline characters

When using the preceding modifiers, the change applies only to the first modifiable word unless the **g** modifier precedes the selected modifier.

If you give a history reference without an event specification (for example, `!$`), the shell uses the previous command as the event. If a previous history reference occurs on the same line, the shell repeats the previous reference. Thus, the following sequence gives the first and last arguments of the command that matches `?foo?`.

```
!?foo?^ !$
```

A special abbreviation of a history reference occurs when the first nonblank character of an input line is a carat (^). This is equivalent to `!:s^`, thus providing a convenient shorthand for substitutions on the text of the previous line. The command `^ lb^` lib corrects the spelling of `lib` in the command.

If necessary, you can enclose a history substitution in braces `{ }` to insulate it from the characters that follow. For example, if you want to use a reference to the command:

```
ls -ld ~paul
```

to perform the command:

```
ls -ld ~paula
```

use the following construction:

```
!{l}a
```

In this example, `!{l}a` looks for a command starting with `l` and appends a to the end.

Quotation with single and double quotes
To prevent further interpretation of all or some of the substitutions, enclose strings in single and double quotation marks.

Enclosing strings in `' '` prevents further interpretation, while enclosing strings in `" "` allows further expansion. In both cases, the text that results becomes all or part of a single word.

Input and output redirection in the C shell

Before the C shell executes a command, it scans the command line for redirection characters. These special notations direct the shell to redirect input and output.

You can redirect the standard input and output of a command with the following syntax statements:

Item	Description
< *File*	Opens the specified *File* (which is first variable, command, and file name expanded) as the standard input.
<<*Word*	Reads the shell input up to the line that matches the value of the *Word* variable. The *Word* variable is not subjected to variable, file name, or command substitution. Each input line is compared to the *Word* variable before any substitutions are done on the line. Unless a quoting character (\, ", ' or `) appears in the *Word* variable, the shell performs variable and command substitution on the intervening lines, allowing the \ character to quote the $, \, and ` characters. Commands that are substituted have all blanks, tabs, and newline characters preserved, except for the final newline character, which is dropped. The resultant text is placed in an anonymous temporary file, which is given to the command as standard input.

Item	Description
> *File* >! *File* >& *File* >&! *File*	Uses the specified *File* as standard output. If *File* does not exist, it is created. If *File* exists, it is truncated, and its previous contents are lost. If the *noclobber* shell variable is set, *File* must not exist or be a character special file, or an error results. This helps prevent accidental destruction of files. In this case, use the forms including a `!` to suppress this check. *File* is expanded in the same way as < input file names. The form >& redirects both standard output and standard error to the specified *File*. The following example shows how to separately redirect standard output to `/dev/tty` and standard error to `/dev/null`. The parentheses are required to allow standard output and standard error to be separate. `% (find / -name vi -print > /dev/tty) >& /dev/null`
> >*File* > >! *File* > >& *File* > >&! *File*	Uses the specified *File* as standard output like >, but *appends* output to the end of *File*. If the *noclobber* shell variable is set, an error results if *File* does not exist, unless one of the forms including a `!` is given. Otherwise, it is similar to >.

A command receives the environment in which the shell was invoked, as changed by the input/output parameters and the presence of the command as a pipeline. Thus, unlike some previous shells, commands that run from a shell script do not have access to the text of the commands by default. Instead, they receive the original standard input of the shell. Use the << mechanism to present inline data, which allows shell command files to function as components of pipelines and also lets the shell block read its input. Note that the default standard input for a command run detached is not changed to the empty `/dev/null` file. Instead, the standard input remains the original standard input of the shell.

To redirect the standard error through a pipe with the standard output, use the form `|&` rather than only the `|`.

Flow control in the C shell
The shell contains commands that can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from shell command-line input. These commands all operate by forcing the shell to repeat, or skip, in its input.

The **foreach**, **switch**, and **while** statements, and the **if-then-else** form of the **if** statement, require that the major keywords appear in a single simple command on an input line.

If the shell input is not searchable, the shell buffers input whenever a loop is being read and searches the internal buffer to do the re-reading implied by the loop. To the extent that this is allowed, backward **goto**s succeed on inputs that you cannot search.

Operating system security

The goal of computer security is the protection of information stored on the computer system.

Information security is aimed at the following:

Item	Description
Integrity	The value of all information depends upon its accuracy. If unauthorized changes are made to data, this data loses some or all of its value.
Privacy	The value of much information depends upon its secrecy.
Availability	Information must be readily available.

It is helpful to plan and implement your security policies before you begin using the system. Security policies are very time-consuming to change later, so up-front planning can save a lot of time later.

Identification and authentication

Identification and authentication establish your identity.

You are required to log in to the system. You supply your user name and a password if the account has one (in a secure system, all accounts should either have passwords or be invalidated). If the password is correct, you are logged in to that account; you acquire the access rights and privileges of the account.

Because the password is the only protection for your account, select and guard your password carefully. Many attempts to break into a system start with attempts to guess passwords. The operating system provides significant password protection by storing user passwords separately from other user information. The encrypted passwords and other security-relevant data for users are stored in the `/etc/security/passwd` file. This file should be accessible only by the root user. With this restricted access to the encrypted passwords, an attacker cannot decipher the password with a program that simply cycles through all possible or likely passwords.

It is still possible to guess passwords by repeatedly attempting to log in to an account. If the password is trivial or is infrequently changed, such attempts might easily succeed.

Login user IDs

The operating system can identify users by their *login user ID*.

The login user ID allows the system to trace all user actions to their source. After a user logs in to the system and before the initial user program is run, the system sets the login ID of the process to the user ID found in the user database. All subsequent processes during the login session are tagged with this ID. These tags provide a trail of all activities performed by the login user ID.

You can reset the *effective user ID*, *real user ID*, *effective group ID*, *real group ID*, and *supplementary group ID* during the session, but you cannot change the login user ID.

Unattended terminals

All systems are vulnerable if terminals are left logged in and unattended. The most serious problem occurs when a system manager leaves a terminal unattended that has been enabled with root authority. In general, users should log out anytime they leave their terminals.

You can force a terminal to log out after a period of inactivity by setting the `TMOUT` and `TIMEOUT` parameters in the `/etc/profile` file. The `TMOUT` parameter works in the **ksh** (Korn) shell, and the `TIMEOUT` parameter works in the **bsh** (Bourne) shell.

The following example, taken from a `.profile` file, forces the terminal to log out after an hour of inactivity:

```
TO=3600
echo "Setting Autologout to $TO"
TIMEOUT=$TO
TMOUT=$TO
export TIMEOUT TMOUT
```

Note: You can override the `TMOUT` and `TIMEOUT` values in the `/etc/profile` file by specifying different values in the `.profile` file in your home directory.

Related concepts

Variable substitution in the Bourne shell
The Bourne shell permits you to perform variable substitutions.

Related reference

Parameter substitution in the Korn shell or POSIX shell

The Korn shell, or POSIX shell, lets you perform parameter substitutions.

File ownership and user groups

Initially, a file's owner is identified by the user ID of the person who created the file.

The owner of a file determines who may read, write (modify), or execute the file. Ownership can be changed with the **chown** command.

Every user ID is assigned to a group with a unique group ID. The system manager creates the groups of users when setting up the system. When a new file is created, the operating system assigns permissions to the user ID that created it, to the group ID containing the file owner, and to a group called `others`, consisting of all other users. The **id** command shows your user ID (UID), group ID (GID), and the names of all groups you belong to.

In file listings (such as the listings shown by the **ls** command), the groups of users are always represented in the following order: user, group, and others. If you need to find out your group name, the **groups** command shows all the groups for a user ID.

Changing file or directory ownership

Use the **chown** command to change the owner of your files.

When you specify the **-R** option, the **chown** command recursively descends through the directory structure from the specified directory. When symbolic links are encountered, the ownership of the file or directory pointed to by the link is changed; the ownership of the symbolic link is not changed.

Note: Only the root user can change the owner of another file. Errors are not displayed when the **-f** option is specified.

For example, to change the owner of the `program.c` file, type the following:

```
chown jim program.c
```

The user-access permissions for the `program.c` file now apply to `jim`. As the owner, `jim` can use the **chmod** command to permit or deny other users access to the `program.c` file.

See the **chown** command for the complete syntax.

File and directory access modes

Every file has an owner. For new files, the user who creates the file is the owner of that file. The owner assigns an *access mode* to the file. Access modes grant other system users permission to read, modify, or execute the file. Only the file's owner or users with root authority can change the access mode of a file.

There are the three classes of users: user/owner, group, and all others. Access is granted to these user classes in some combination of three modes: read, write, or execute. When a new file is created, the default permissions are read, write, and execute permission for the user who created the file. The other two groups have read and execute permission. The following table illustrates the default file-access modes for the three classes of user groups:

Item	**Description**		
Classes	**Read**	**Write**	**Execute**
Owner	Yes	Yes	Yes
Group	Yes	No	Yes
Others	Yes	No	Yes

The system determines who has permission and the level of permission they have for each of these activities. Access modes are represented both symbolically and numerically in the operating system.

Related concepts

Types of files

The types of files recognized by the system are either **regular**, **directory**, or **special**. However, the operating system uses many variations of these basic types.

Symbolic representation of access modes
Access modes are represented symbolically.

Item	Description
r	Indicates read permission, which allows users to view the contents of a file.
w	Indicates write permission, which allows users to modify the contents of a file.
x	Indicates execute permission. For executable files (ordinary files that contain programs), execute permission means that the program can be run. For directories, execute permission means the contents of the directory can be searched.

The access modes for files or directories are represented by nine characters. The first three characters represent the current **Owner** permissions, the second sent of three characters represents the current **Group** permissions, and the third set of three characters represents the current settings for the **Other** permissions. A hyphen (**-**) in the nine-character set indicates that no permission is given. For example, a file with the access modes set to `rwxr-xr-x` gives read and execute permission to all three groups and write permission only to the owner of the file. This is the symbolic representation of the default setting.

The **ls** command, when used with the **-l** (lower case L) flag, gives a detailed listing of the current directory. The first 10 characters in the **ls -l** listing show the file type and permissions for each of the three groups. The **ls -l** command also lists the owner and group associated with each file and directory.

The first character indicates the type of file. The remaining nine characters contain the file permission information for each of the three classes of users. The following symbols are used to represent the type of file:

Item	Description
-	Regular files
d	Directory
b	Block special files
c	Character special files
p	Pipe special files
l	Symbolic links
s	Sockets

For example, this is a sample **ls -l** listing:

```
-rwxrwxr-x   2   janet   acct   512 Mar 01 13:33 january
```

Here, the first hyphen (-) indicates a regular file. The next nine characters (`rwxrwxr-x` represent the User, Group, and Other access modes, as discussed above. `janet` is the file owner, and `acct` is the name of Janet's group. 512 is the file size in bytes, `Mar 01 13:33` is the last date and time of modification, and `january` is the file name. The 2 indicates how many links exist to the file.

Numeric representation of access modes
Numerically, read access is represented by a value of 4, write permission is represented by a value of 2, and execute permission is represented by a value of 1. The total value between 1 and 7 represents the access mode for each group (user, group, and other).

The following table illustrates the numeric values for each level of access:

Total Value	Read	Write	Execute
0	-	-	-
1	-	-	1
2	-	2	-
3	-	2	1
4	4	-	-
5	4	-	1
6	4	2	-
7	4	2	1

When a file is created, the default file access mode is 755. This means the user has read, write, and execute permissions (4+2+1=7), the group has read and execute permission (4+1=5), and all others have read and execute permission (4+1=5). To change access permission modes for files you own, run the **chmod** (change mode) command.

Displaying group information

Use the **lsgroup** command to display the attributes of all the groups on the system (or of specified groups). If one or more attributes cannot be read, the **lsgroup** command lists as much information as possible.

The attribute information displays as *Attribute*=*Value* definitions, each separated by a blank space.

1. To list all of the groups on the system, type the following:

   ```
   lsgroup ALL
   ```

   The system displays each group, group ID, and all of the users in the group in a list similar to the following:

   ```
   system  0       arne,pubs,ctw,geo,root,chucka,noer,su,dea,backup,build,janice,denise
   staff   1       john,ryan,flynn,daveb,jzitt,glover,maple,ken,gordon,mbrady
   bin     2       root,bin
   sys     3       root,su,bin,sys
   ```

2. To display specific attributes for all groups, do either of the following:

   - You can list attributes in the form *Attribute*=*Value* separated by a blank space. This is the default style. For example, to list the ID and users for all of the groups on the system, type the following:

     ```
     lsgroup -a id users ALL | pg
     ```

     A list similar to the following is displayed:

     ```
     system id=0 users=arne,pubs,ctw,geo,root,chucka,noer,su,dea,backup,build
     staff id=1 users=john,ryan,flynn,daveb,jzitt,glover,maple,ken
     ```

   - You can also list the information in stanza format. For example, to list the ID and users for all of the groups on the system in stanza format, type the following:

     ```
     lsgroup -a -f id users ALL | pg
     ```

     A list similar to the following is displayed:

     ```
     system:
        id=0
        users=pubs,ctw,geo,root,chucka,noer,su,dea,backup,build

     staff:
        id=1
        users=john,ryan,flynn,daveb,jzitt,glover,maple,ken
     ```

```
bin:
   id=2
   users=root,bin

sys:
   id=3
   users=root,su,bin,sys
```

3. To display all attributes for a specific group, you can use one of two styles for listing specific attributes for all groups:
   - You can list each attribute in the form *Attribute=Value* separated by a blank space. This is the default style. For example, to list all attributes for the group system, type the following:

     ```
     lsgroup system
     ```

     A list similar to the following is displayed:

     ```
     system id=0 users=arne,pubs,ctw,geo,root,chucka,noer,su,dea,backup,build,janice,denise
     ```

   - You can also list the information in stanza format. For example, to list all attributes for the group `bin` in stanza format, type the following:

     ```
     lsgroup -f system
     ```

     A list similar to the following is displayed:

     ```
     system:
        id=0   users=arne,pubs,ctw,geo,root,chucka,noer,su,dea,backup,build,janice,denise
     ```

4. To list specific attributes for a specific group, type the following:

   ```
   lsgroup -a Attributes Group
   ```

   For example, to list the ID and users for group `bin`, type the following:

   ```
   lsgroup -a id users bin
   ```

   A list similar to the following is displayed:

   ```
   bin id=2 users=root,bin
   ```

See the **lsgroup** command for the complete syntax.

Changing file or directory permissions

Use the **chmod** command to change the permissions of your files.

1. To add a type of permission to the `chap1` and `chap2` files, type the following:

   ```
   chmod g+w chap1 chap2
   ```

   This adds write permission for group members to the files `chap1` and `chap2`.
2. To make several permission changes at once to the `mydir` directory, type the following:

   ```
   chmod go-w+x mydir
   ```

   This denies (**-**) group members (**g**) and others (**o**) the permission to create or delete files (**w**) in the `mydir` directory and allows (**+**) group members and others to search the `mydir` directory or use (**x**) it in a path name. This is equivalent to the following command sequence:

   ```
   chmod g-w mydir
   chmod o-w mydir
   chmod g+x mydir
   chmod o+x mydir
   ```

3. To permit only the owner to use a shell procedure named **cmd** as a command, type the following:

```
chmod u=rwx,go= cmd
```

This gives read, write, and execute permission to the user who owns the file (**u=rwx**). It also denies the group and others the permission to access `cmd` in any way (**go=**).

4. To use the numeric mode form of the **chmod** command to change the permissions of the `text`, file type the following:

```
chmod 644 text
```

This sets read and write permission for the owner, and it sets read-only mode for the group and others.

See the **chmod** command for the complete syntax.

Access control lists

Access control consists of protected information resources that specify who can be granted access to such resources.

The operating system allows for need-to-know or discretionary security. The owner of an information resource can grant other users read or write access rights for that resource. A user who is granted access rights to a resource can transfer those rights to other users. This security allows for user-controlled information flow in the system; the owner of an information resource defines the access permissions to the object.

Users have user-based access only to the objects that they own. Typically, users receive either the group permissions or the default permissions for a resource. The major task in administering access control is to define the group memberships of users, because these memberships determine the users' access rights to the files that they do not own.

Access control lists for file system objects

File system objects are typically associated with an Access Control List (ACL), which normally consists of series of Access Control Entries (ACEs). Each ACE defines the identity and its related access rights.

To maintain access control lists, use the **aclget**, **acledit**, **aclput** and **aclconvert** commands.

Note that ACL is typically stored and managed on the media by the physical file system (PFS). The AIX operating system provides an infrastructure for physical file systems to support and manage multiple ACL types. The JFS2 file system shipped with AIX supports two ACL types:

- AIXC
- NFS4

Earlier file systems supported only the AIXC ACL type as in the previous AIX releases. These ACL types are discussed in detail in the Security.

AIXC access control list type

The AIXC (AIX Classic) ACL type provides for the ACL behavior as defined on previous releases of AIX. This ACL type consists of the regular base mode bits and extended permissions (ACEs).

With extended permissions, you can permit or deny file access to specific individuals or groups without changing the base permissions.

Note: The AIXC ACL for a file cannot exceed one memory page (approximately 4096 bytes) in size.

The **chmod** command in numeric mode (with octal notations) can set base permissions and attributes. The `chmod` subroutine, which the command calls, disables extended permissions. Extended permissions are disabled if you use the numeric mode of the **chmod** command on a file that has an ACL. The symbolic mode of the **chmod** command does not disable extended permissions when the ACL associated is of type AIXC. For more information on numeric and symbolic mode, refer to the **chmod** command. For information about the chmod command, see **chmod**.

Base permissions

AIXC ACL specific base permissions are the traditional file-access modes assigned to the file owner, file group, and other users. The access modes are read (`r`), write (`w`), and execute/search (`x`).

Note: AIXC ACL type Base Permissions will be same as the file mode bits stored in the file system object's inode headers. That is, the information in base mode bits is same as the value returned by file system when **`stat`** is performed on the file system object.

In an access control list, base permissions are in the following format, with the **Mode** parameter expressed as `rwx` (with a hyphen (-) replacing each unspecified permission):

```
base permissions:
   owner(name): Mode
   group(group): Mode
   others: Mode
```

Attributes

Three attributes can be added to an access control list:

setuid (SUID)
Set-user-ID mode bit. This attribute sets the effective and saved user IDs of the process to the owner ID of the file on execution.

setgid (SGID)
Set-group-ID mode bit. This attribute sets the effective and saved group IDs of the process to the group ID of the file on execution.

savetext (SVTX)
Saves the text in a text file format.

The above attributes are added in the following format:

```
attributes: SUID, SGID, SVTX
```

Extended permissions

AIXC ACL extended permissions allow the owner of a file to more precisely define access to that file. Extended permissions modify the base file permissions (owner, group, others) by permitting, denying, or specifying access modes for specific individuals, groups, or user and group combinations. Permissions are modified through the use of keywords.

The permit, deny, and specify keywords are defined as follows:

permit
Grants the user or group the specified access to the file

deny
Restricts the user or group from using the specified access to the file

specify
Precisely defines the file access for the user or group

If a user is denied a particular access by either a deny or a specify keyword, no other entry can override that access denial.

The enabled keyword must be specified in the ACL for the extended permissions to take effect. The default value is the disabled keyword.

In an AIXC ACL, extended permissions are in the following format:

```
extended permissions:
  enabled | disabled
    permit   Mode  UserInfo...:
    deny     Mode  UserInfo...:
    specify  Mode  UserInfo...:
```

Use a separate line for each permit, deny, or specify entry. The **Mode** parameter is expressed as `rwx` (with a hyphen (-) replacing each unspecified permission). The **`UserInfo`** parameter is

expressed as `u:UserName`, or `g:GroupName`, or a comma-separated combination of `u:UserName` and `g:GroupName`.

Note: If more than one user name is specified in an entry, that entry cannot be used in an access control decision because a process has only one user ID.

NFS4 access control list type
JFS2 file system in AIX also supports NFS4 ACL type. This ACL implementation follows the ACL definition as specified in NFS4 version 4 protocol related RFC.

This ACL provides much finer granular control over the access rights and also provides for features such as inheritance. NFS4 ACL consists of an array of ACEs. Each ACE defines access rights for an identity. As defined in the RFC, the main components of NFS4 ACE are as follows:

```
struct nfsace4 {

                 acetype4        type;
                 aceflag4        flag;
                 acemask4        access_mask;
                 utf8str_mixed   who;
};
```

Where:

type
Bit mask that defines the type of the ACE. Details such as whether this ACE allows access or denies access are defined here.

flag
Bit mask that describes the inheritance aspects of the ACE. Defines whether this ACE is applicable to the file system object, or its children, or both.

access_mask
Bit mask that defines various access rights possible. Rights defined include, read, write, execute, create, delete, create child, delete child, etc.

who
This null-terminated string defines the identity of the person to which this ACE will apply. Note that per RFC, the size of this string is unlimited, and a loose definition allows for defining domains within NFS version 4 networks to manage access control. Natively (most of the time) AIX does not interpret this string, and each ACE is associated with an AIX-understood identity (such as `uid` or `gid`). It is expected that the NFS version 4 file system will interpret these strings as necessary to convert them to OS-understood user or group IDs. AIX only understands some of the special `who` strings defined in the RFC.

In AIX, use the **aclget**, **acledit**, **aclput** and **aclconvert** commands to manage NFS4 ACLs.

Note: Any type of **chmod** command will erase the file's ACL.

Access control list example for AIXC
The following is an example of an AIXC access control list (ACL).

The following is an example of an AIXC ACL:

```
attributes: SUID
base permissions:
      owner(frank):  rw-
      group(system): r-x
      others: ---
extended permissions:
      enabled
        permit  rw-  u:dhs
        deny    r--  u:chas, g:system
        specify r--  u:john, g:gateway, g:mail
        permit  rw-  g:account, g:finance
```

The parts of the ACL and their meanings are as follows:

- The first line indicates that the setuid bit is turned on.

- The next line, which introduces the base permissions, is optional.
- The next three lines specify the base permissions. The owner and group names in parentheses are for information only. Changing these names does not alter the file owner or file group. Only the **chown** command and the **chgrp** command can change these file attributes.
- The next line, which introduces the extended permissions, is optional.
- The next line indicates that the extended permissions that follow are enabled.
- The last four lines are the extended entries.
- The first extended entry grants user `dhs` read (`r`) and write (`w`) permission on the file.
- The second extended entry denies read (`r`) access to user `chas` only when he is a member of the system group.
- The third extended entry specifies that as long as user `john` is a member of both the gateway group and the mail group, this user has read (`r`) access. If user `john` is not a member of both groups, this extended permission does not apply.
- The last extended entry grants any user in **both** the `account` group and the `finance` group read (`r`) and write (`w`) permission.

Note: More than one extended entry can be applied to a process, with restrictive modes taking precedence over permissive modes.

See the **acledit** command for the complete syntax.

Access control list access authorization

The owner of the information resource is responsible for managing access rights. Resources are protected by permission bits, which are included in the mode of the object.

For AIXC ACL, the permission bits define the access permissions granted to the owner of the object, the group of the object, and for the `others` default class. AIXC ACL type supports three different modes of access (read, write, and execute) that can be granted separately.

When a user logs in to an account (using the **login** or **su** command), the user IDs and group IDs assigned to that account are associated with the user's processes. These IDs determine the access rights of the process.

For files, directories, named pipes, and devices (special files) with an associated AIX ACL, access is authorized as follows:

- For each access control entry (ACE) in the access control list (ACL), the identifier list is compared to the identifiers of the process. If there is a match, the process receives the permissions and restrictions defined for that entry. The logical unions for both permissions and restrictions are computed for each matching entry in the ACL. If the requesting process does not match any of the entries in the ACL, it receives the permissions and restrictions of the default entry.
- If the requested access mode is permitted (included in the union of the permissions) and is not restricted (included in the union of the restrictions), access is granted. Otherwise, access is denied.

Further, for an AIXC ACL type, the identifier list of an ACL matches a process if all identifiers in the list match the corresponding type of effective identifier for the requesting process. A USER-type identifier matched is equal to the effective user ID of the process, and a GROUP-type identifier matches if it is equal to the effective group ID of the process or to one of the supplementary group IDs. For instance, an ACE with an identifier list such as the following:

```
USER:fred, GROUP:philosophers, GROUP:software_programmer
```

would match a process with an effective user ID of `fred` and a group set of:

```
philosophers, philanthropists, software_programmer, doc_design
```

but would not match for a process with an effective user ID of `fred` and a group set of:

```
philosophers, iconoclasts, hardware_developer, graphic_design
```

Note that an ACE with an identifier list of the following would match for both processes:

```
USER:fred, GROUP:philosophers
```

In other words, the identifier list in the ACE functions is a set of conditions that must hold for the specified access to be granted.

The discretionary access control mechanism allows for effective access control of information resources and provides for separate protection of the confidentiality and integrity of the information. Owner-controlled access control mechanisms are only as effective as users make them. All users must understand how access permissions are granted and denied, and how these are set.

Note that file system objects with an associated NFS4 ACL type, access checks are based on various ACEs that form the ACL as per the rules setup in the NFS version 4 protocol-related RFC. Identity matching is done based on the user ID or group ID or special who strings defined in the ACE against the process's credentials. If a match occurs, the access rights requested are checked against the access rights defined in the ACE. If any of the access rights are allowed, those will be taken out, and the compare operation continues on to the next ACE. This process is continued until either the ACL end is reached, or all the access rights are met, or if any of the access rights requested are denied. The following steps capture the access checking in the case of a file system object with an associated NFS4 ACL:

1. For each access control entry (ACE) in the access control list (ACL), the identifier list is compared to the identifiers of the process. Identity checks include the user ID or group ID defined in the ACE. Also, if the identity is defined as `special` with strings such as `OWNER@`, a match will occur if the calling process is by the owner of the file. If there is a match, the process receives the access rights defined for that entry. Else, continue to the next ACE.
2. Requested access rights are compared with the access rights retrieved from ACE entry. If any of the access rights requested are explicitly denied by the ACE, then the access checking process is ended, and the requesting process will be denied access.
3. If some of the requested access rights are met by the ACE, then those access rights will be taken out from the requests access rights list, and the compare operation continues to the next ACE.
4. If all of the requested access rights are met by the ACEs, then the requested access is allowed.
5. If ACL end is reached before all of the requested access rights are resolved, then the access is denied.

Note that apart from the ACL type-based access checks, individual physical file systems might also choose to provide for privilege-based access to the file system objects. For example, an owner might always at least have the permission to modify the ACL, irrespective of the existing ACL access rights. A process with a user ID of 0 is known as a root user process. These processes are generally allowed all access permissions. However, if a root user process requests execute permission for a program, access is granted only if execute permission is granted to at least one user.

All access permission checks for these objects are made at the system call level when the object is first accessed. Because System V Interprocess Communication (SVIPC) objects are accessed statelessly, checks are made for every access. However, it is possible that checks are made by the physical file systems at open time of the file system object and not at the time of read or write operation. For objects with file system names, it is necessary to be able to resolve the name of the actual object. Names are resolved either relatively (to the process' working directory) or absolutely (to the process' root directory). All name resolution begins by searching one of these.

Command for displaying access control information (aclget command)

The **aclget** command displays the access control information of a file. The information that you view includes attributes, base permissions, and extended permissions.

For example, to display the access control information for the `status` file, type the following:

```
aclget status
```

The access control information that displays includes a list of attributes, base permissions, and extended permissions.

Related concepts

Access control list example and description

The following is an example and description of access control lists (ACLs).

Setting access control information (aclput command)

To set the access control information for a file, use the **aclput** command.

Note: The access control list for a file cannot exceed one memory page (approximately 4096 bytes) in size.

See the following examples:

For example, to set the access control information for the status file with the access control information stored in the `acldefs` file, type the following:

```
aclput -i acldefs status
```

To set the access control information for the status file with the same information used for the `plans` file, type the following:

```
aclget plans | aclput status
```

Access control list example and description

The following is an example and description of access control lists (ACLs).

The following is an example of an ACL:

```
attributes: SUID
base permissions:
      owner(frank):  rw-
      group(system): r-x
      others: ---
extended permissions:
      enabled
        permit  rw-  u:dhs
        deny    r--  u:chas, g:system
        specify r--  u:john, g:gateway, g:mail
        permit  rw-  g:account, g:finance
```

The parts of the ACL and their meanings are the following:

- The first line indicates that the **setuid** bit is turned on.
- The next line, which introduces the base permissions, is optional.
- The next three lines specify the base permissions. The owner and group names in parentheses are for information only. Changing these names does not alter the file owner or file group. Only the **chown** command and the **chgrp** command can change these file attributes.
- The next line, which introduces the extended permissions, is optional.
- The next line indicates that the extended permissions that follow are enabled.
- The last four lines are the extended entries. The first extended entry grants user `dhs` read (`r`) and write (`w`) permission on the file.
- The second extended entry denies read (`r`) access to user `chas` only when he is a member of the `system` group.
- The third extended entry specifies that as long as user `john` is a member of both the `gateway` group and the `mail` group, has read (`r`) access. If user `john` is not a member of both groups, this extended permission does not apply.
- The last extended entry grants any user in **both** the `account` group and the `finance` group read (`r`) and write (`w`) permission.

  Note: More than one extended entry can be applied to a process, with restrictive modes taking precedence over permissive modes.

  See the **acledit** command for the complete syntax.

Related concepts

Command for displaying access control information (aclget command)
The **aclget** command displays the access control information of a file. The information that you view includes attributes, base permissions, and extended permissions.

Related tasks

Editing access control information (acledit command)
Use the **acledit** command to change the access control information of a file. The command displays the current access control information and lets the file owner change it.

Editing access control information (acledit command)

Use the **acledit** command to change the access control information of a file. The command displays the current access control information and lets the file owner change it.

Before making any permanent changes, the **acledit** command asks if you want to proceed.

Note: The *EDITOR* environment variable must be specified with a complete path name; otherwise, the **acledit** command will fail.

The access control information that displays is ACL type specific and includes a list of attributes, base permissions, and extended permissions.

For example, to edit the access control information of the `plans` file, type the following:

```
acledit plans
```

Related concepts

Access control list example and description
The following is an example and description of access control lists (ACLs).

Locking your terminal (lock or xlock command)

Use the **lock** command to lock your terminal. The **lock** command requests your password, reads it, and requests the password a second time to verify it.

In the interim, the command locks the terminal and does not relinquish it until the password is received the second time. The timeout default value is 15 minutes, but this can be changed with the **-***Number* flag.

Note: If your interface is AIXwindows, use the **xlock** command in the same manner.

For example, to lock your terminal under password control, type the following:

```
lock
```

You are prompted for the password twice so the system can verify it. If the password is not repeated within 15 minutes, the command times out.

To reserve a terminal under password control with a timeout interval of 10 minutes, type the following:

```
lock -10
```

Authentication

The **xlock** command is a Pluggable Authentication Module (PAM) enabled X server command that locks the X server until the user enters a password. It supports both local UNIX authentication and PAM authentication for unlocking the X server.

You can set the system-wide configuration to use PAM for authentication by providing root user access and by modifying the value of the *auth_type* attribute to *PAM_AUTH* in the **usw** stanza of the `/etc/security/login.cfg` file.

The authentication mechanisms that are used when PAM is enabled are dependent on the configuration of the login service in the `/etc/pam.conf` file. The **xlock** command requires

the /etc/pam.conf file entry for the **auth**, **account**, **password**, and **session** module types. The following configuration is recommended for the /etc/pam.conf file entry in the **xlock** command:

```
xlock               auth                required          pam_aix
xlock               account             required          pam_aix
xlock               password            required          pam_aix
xlock               session             required          pam_aix
```

Command summary for file and system security

The following are commands for file system and security.

Item	Description
acledit	Edits the access control information of a file
aclget	Displays the access control information of a file
aclput	Sets the access control information of a file
chmod	Changes permission modes
chown	Changes the user associated with a file
lock	Reserves a terminal
lsgroup	Displays the attributes of groups
xlock	Locks the local X display until a password is entered

User environment

Each login name has its own system environment.

The system environment is an area where information that is common to all processes running in a session is stored. You can use several commands to display information about your system.

User environment files and customization procedures

These files and procedures help the user customize the system environment.

System startup files

Item	Description
/etc/profile	System file that contains commands that the system executes when you log in.
/etc/environment	System file that contains variables specifying the basic environment for all processes.
$HOME/.profile	File in your home directory that contains commands that override the system **/etc/profile** when you log in. For more information, see .profile file.
$HOME/.env	File in your home directory that overrides the system **/etc/environment** and contains variables specifying the basic environment for all processes. For more information, see .env file.

AIXwindows startup files

Item	Description
$HOME/.xinitrc	File in your home directory that controls the windows and applications that start up when you start AIXwindows. For more information, see .xinitrc file.
$HOME/.Xdefaults	File in your home directory that controls the visual or behavioral aspect of AIXwindows resources. For more information, see ".Xdefaults file" on page 323.
$HOME/.mwmrc	File in your home directory that defines key bindings, mouse button bindings, and menu definitions for your window manager. For more information, see ".mwmrc file" on page 323.

Customization procedures

Item	Description
PS1	Normal system prompt
PS2	More input system prompt
PS3	Root system prompt
chfont	Changes the font used by a display at system restart
stty	Sets, resets, and reports workstation operating parameters

System devices list (lscfg command)

To display the name, location, and description of each device found in the current configuration, use the **lscfg** command. The list is sorted by device location.

For example, to list the devices configured in your system, at the prompt, type the following:

```
lscfg
```

The system displays output similar to the following:

```
INSTALLED RESOURCE LIST

The following resources are installed on your machine.

+/- = Added/Deleted from Diagnostic Test List.
*   = NOT Supported by Diagnostics.


  Model Architecture: chrp
  Model Implementation: Multiple Processor, PCI bus

+  sysplanar0   00-00           CPU Planar
+  fpa0         00-00           Floating Point Processor
+  mem0         00-0A           Memory Card
+  mem1         00-0B           Memory Card
+  ioplanar0    00-00           I/O Planar
+  rs2320       00-01           RS232 Card
+  tty0         00-01-0-01      RS232 Card Port
-  tty1         00-01-0-02      RS232 Card Port
   ..
   ..
   ..
```

The device list is not sorted by device location alone. It is sorted by the parent/child hierarchy. If the parent has multiple children, the children are sorted by device location. If the children have the same device location, they are displayed in the order in which they were obtained by the software. To display

information about a specific device, you can use the **-l** flag. For example, to list the information on device **sysplanar0**, at the prompt, type the following:

```
lscfg -l sysplanar0
```

The system displays output similar to the following:

```
DEVICE          LOCATION     DESCRIPTION

sysplanar0      00-00        CPU Planar
```

You can also use the **lscfg** command to display vital product data (VPD), such as part numbers, serial numbers, and engineering change levels. For some devices, the VPD is collected automatically and added to the system configuration. For other devices, the VPD is typed manually. An ME preceding the data indicates that the data was typed manually.

For example, to list VPD for devices configured in your system, at the prompt, type the following:

```
lscfg -v
```

The system displays output similar to the following:

```
INSTALLED RESOURCE LIST WITH VPD
```

```
The following resources are installed in your machine.
```

```
  Model Architecture: chrp
  Model Implementation: Multiple Processor, PCI bus
```

```
sysplanar0   00-00   CPU Planar
```

```
   Part Number.........342522
   EC Level............254921
   Serial Number.......353535
```

```
fpa0   00-00   Floating Point Processor
mem0   00-0A   Memory Card
```

```
   EC Level............990221
.
.
.
```

Displaying console names

To write the name of the current console device to standard output (usually your screen), use the **lscons** command.

For example, at the prompt, type the following:

```
lscons
```

The system displays output similar to the following:

```
/dev/lft0
```

See the **lscons** command for the complete syntax.

Displaying the terminal name (tty command)

To display the name of your terminal, use the **tty** command.

For example, at the prompt, type the following:

```
tty
```

The system displays information similar to the following:

```
/dev/tty06
```

In this example, `tty06` is the name of the terminal, and **/dev/tty06** is the device file that contains the interface to this terminal.

Listing available displays (lsdisp command)

To list the displays currently available on your system, providing a display identification name, slot number, display name, and description of each of the displays, use the **lsdisp** command.

For example, to list all available displays, type the following:

```
lsdisp
```

The following is an example of the output. The list displays in ascending order according to slot number.

```
Name   Slot    Name        Description
ppr0   00-01   POWER_G4    Midrange Graphics Adapter
gda0   00-03   colorgda    Color Graphics Display Adapter
ppr1   00-04   POWER_Gt3   Midrange Entry Graphics Adapter
```

Listing available fonts (lsfont command)

To display a list of the fonts available to your display, use the **lsfont** command.

For example, to list all fonts available to the display, type the following:

```
lsfont
```

The following is an example of the output, showing the font identifier, file name, glyph size, and font encoding:

```
FONT  FILE            GLYPH  FONT
ID    NAME            SIZE   ENCODING
====  ==============  =====  =========
0     Erg22.iso1.snf  12x30  ISO8859-1
1     Erg11.iso1.snf   8x15  ISO8859-1
```

Listing the current software keyboard map (lskbd command)

To display the absolute path name of the current software keyboard map loaded into the system, use the **lskbd** command.

For example, to list your current keyboard map, type the following:

```
lskbd
```

The following is an example of the listing displayed by the **lskbd** command:

```
The current software keyboard map = /usr/lib/nls/loc/C.lftkeymap
```

Listing available software products (lslpp command)

To display information about software products available on your system, use the **lslpp** command.

For example, to list all the software products in your system, at the system prompt, type the following:

```
lslpp -l -a
```

The following is an example of the output:

```
Fileset                 Level  State     Description
--------------------  ------- --------  -----------------
Path: /usr/lib/objrepos
   X11_3d.gl.dev.obj            APPLIED   AIXwindows/3D GL
                                          Development Utilities
Fonts
  X11fnt.oldX.fnt               APPLIED   AIXwindows Miscellaneous
                                          X Fonts
X11mEn_US.msg                   APPLIED   AIXwindows NL Message
                                          files
```

```
.
```

```
.
```

```
.
```

If the listing is very long, the top portion may scroll off the screen. To display the listing one page (screen) at a time, use the **lslpp** command piped to the **pg** command. At the prompt, type the following:

```
lslpp -l -a | pg
```

List of control key assignments for your terminal (stty command)

To display your terminal settings, use the **stty** command. Note especially which keys your terminal uses for control keys.

For example, at the prompt, type the following:

```
stty -a
```

The system displays information similar to the following:

```
.
```

```
.
```

```
.
```

```
intr = ^C; quit = ^\; erase = ^H; kill = ^U; eof = ^D;
eol = ^@ start = ^Q; stop = ^S; susp = ^Z; dsusp = ^Y;
reprint = ^R discard = ^O; werase = ^W; lnext = ^V
```

```
.
```

```
.
```

```
.
```

In this example, lines such as `intr = ^C; quit = ^\; erase = ^H;` display your control key settings. The ^H key is the Backspace key, and it is set to perform the erase function.

If the listing is very long, the top portion may scroll off the screen. To display the listing one page (screen) at a time, use the **stty** command piped to the **pg** command. At the prompt, type the following:

```
stty -a | pg
```

Related concepts

Foreground process cancellation

If you start a foreground process and then decide that you do not want it to finish, you can cancel it by pressing INTERRUPT. This is usually Ctrl-C or Ctrl-Backspace.

Listing environment variables (env command)

To display your current environment variables, use the **env** command. An environment variable that is accessible to all your processes is called a *global variable*.

For example, to list all environment variables and their associated values, type the following:

```
env
```

The following is an example of the output:

```
TMPDIR=/usr/tmp
myid=denise
LANG=En_US
UNAME=barnard
PAGER=/bin/pg
VISUAL=vi
PATH=/usr/ucb:/usr/lpp/X11/bin:/bin:/usr/bin:/etc:/u/denise:/u/denise/bin:/u/bin1
MAILPATH=/usr/mail/denise?denise has mail !!!
MAILRECORD=/u/denise/.Outmail
EXINIT=set beautify noflash nomesg report=1 showmode showmatch
EDITOR=vi
PSCH=>
HISTFILE=/u/denise/.history
LOGNAME=denise
MAIL=/usr/mail/denise
PS1=denise@barnard:${PWD}>
PS3=#
PS2=>
epath=/usr/bin
USER=denise
SHELL=/bin/ksh
HISTSIZE=500
HOME=/u/denise
FCEDIT=vi
TERM=lft
MAILMSG=**YOU HAVE NEW MAIL. USE THE mail COMMAND TO SEE YOUR PWD=/u/denise
ENV=/u/denise/.env
```

If the listing is very long, the top portion scrolls off the screen. To display the listing one page (screen) at a time, use the **env** command piped to the **pg** command. At the prompt, type the following:

```
env | pg
```

Displaying an environment variable value (printenv command)

To display the values of environment variables, use the **printenv** command.

If you specify the **Name** parameter, the system only prints the value associated with the variable you requested. If you do not specify the **Name** parameter, the **printenv** command displays all current environment variables, showing one **Name** =*Value* sequence per line.

For example, to find the current setting of the *MAILMSG* environment variable, type the following:

```
printenv MAILMSG
```

The command returns the value of the *MAILMSG* environment variable. For example:

```
YOU HAVE NEW MAIL
```

Bidirectional languages (aixterm command)

The **aixterm** command supports Arabic and Hebrew, which are bidirectional languages.

Bidirectional languages have the ability to be read and written in two directions: from left to right and from right to left. You can work with Arabic and Hebrew applications by opening a window specifying an Arabic or Hebrew locale.

Command summary for user environment and system information

The following are commands for user environment and system information.

Item	Description
aixterm	Enables you work with bidirectional languages
env	Displays the current environment or sets the environment for the execution of a command
lscfg	Displays diagnostic information about a device
lscons	Displays the name of the current console
lsdisp	Lists the displays currently available on the system
lsfont	Lists the fonts available for use by the display
lskbd	Lists the keyboard maps currently loaded in the system
lslpp	Lists software products
printenv	Displays the values of environment variables
stty	Displays system settings
tty	Displays the full path name of your terminal

User environment customization

The operating system provides various commands and initialization files that enable you to customize the behavior and the appearance of your user environment.

You can also customize some of the default resources of the applications you use on your system. Defaults are initiated by the program at startup. When you change the defaults, you must exit and then restart the program for the new defaults take effect.

For information about customizing the behavior and appearance of the Common Desktop Environment, see the *Common Desktop Environment 1.0: Advanced User's and System Administrator's Guide*.

System startup files

When you log in, the shell defines your user environment after reading the initialization files that you have set up. The characteristics of your user environment are defined by the values given to your environment variables. You maintain this environment until you log out of the system.

The shell uses two types of profile files when you log in to the operating system. It evaluates the commands contained in the files and then executes the commands to set up your system environment. The files have similar functions, except that the `/etc/profile` file controls profile variables for all users on a system, whereas the `.profile` file allows you to customize your own environment.

The shell first runs the commands to set up your system environment in the `/etc/environment` file and then evaluates the commands contained in the `/etc/profile` file. After these files are run, the system then checks to see if you have a `.profile` file in your home directory. If the `.profile` file exists, the system runs this file. The `.profile` file will specify if an environment file also exists. If an environment file exists (usually named `.env`), the system then runs this file and sets up your environment variables.

The `/etc/environment`, `/etc/profile`, and `.profile` files are run once at login time. The `.env` file, on the other hand, is run every time you open a new shell or a window.

/etc/environment file
The first file that the operating system uses at login time is the `/etc/environment` file. The `/etc/environment` file contains variables specifying the basic environment for all processes.

When a new process begins, the `exec` subroutine makes an array of strings available that have the form *Name=Value*. This array of strings is called the *environment*. Each name defined by one of the strings is called an *environment variable* or *shell variable*. The `exec` subroutine allows the entire environment to be set at one time.

When you log in, the system sets environment variables from the `/etc/environment` file before reading your login profile, named `.profile`. The following variables make up the basic environment:

Item	Description
HOME	The full path name of the user's login or HOME directory. The `login` program sets this to the name specified in the `/etc/passwd` file.
LANG	The locale name currently in effect. The *LANG* variable is initially set in the `/etc/profile` file at installation time.
NLSPATH	The full path name for message catalogs.
LOCPATH	The full path name of the location of National Language Support tables.
PATH	The sequence of directories that commands, such as **sh**, **time**, **nice** and **nohup,** search when looking for a command whose path name is incomplete.
TZ	The time zone information. The *TZ* environment variable is initially set by the `/etc/profile` file, the system login profile.

For detailed information about the `/etc/environment` file, see the *Files Reference*.

/etc/profile file
The second file that the operating system uses at login time is the `/etc/profile` file.

The `/etc/profile` file controls system-wide default variables, such as:

- Export variables
- File creation mask (umask)
- Terminal types
- Mail messages to indicate when new mail has arrived

The system administrator configures the `/etc/profile` file for all users on the system. Only the system administrator can change this file.

The following example is a typical `/etc/profile` file:

```
#Set file creation mask
umask 022
#Tell me when new mail arrives
MAIL=/usr/mail/$LOGNAME
#Add my /bin directory to the shell search sequence
PATH=/usr/bin:/usr/sbin:/etc::
#Set terminal type
TERM=lft
#Make some environment variables global
export MAIL PATH TERM
```

For detailed information about the `/etc/profile` file, see the *Files Reference* .

.profile file
The `.profile` file is present in your home ($HOME) directory and lets you customize your individual working environment.

Because the `.profile` file is hidden, use the **ls -a** command to list it.

After the **login** program adds the *LOGNAME* (login name) and *HOME* (login directory) variables to the environment, the commands in the $HOME/.profile file are executed if the file is present. The .profile file contains your individual profile that overrides the variables set in the /etc/profile file. The .profile file is often used to set exported environment variables and terminal modes. You can customize your environment by modifying the .profile file. Use the .profile file to control the following defaults:

- Shells to open
- Prompt appearance
- Keyboard sound

The following example is a typical .profile file:

```
PATH=/usr/bin:/etc:/home/bin1:/usr/lpp/tps4.0/user::
epath=/home/gsc/e3:
export PATH epath
csh
```

This example has defined two path variables (*PATH* and *epath*), exported them, and opened a C shell (csh).

You can also use the .profile file (or if it is not present, the /etc/profile file) to determine login shell variables. You can also customize other shell environments. For example, use the .cshrc file and .kshrc file to customize a C shell and a Korn shell, respectively, when each type of shell is started.

.env file
A fourth file that the operating system uses at login time is the .env file, if your .profile contains the following line: export ENV=$HOME/.env

The .env file lets you customize your individual working environment variables. Because the .env file is hidden, use the **ls -a** command to list it. For more information about the **ls** command, see **ls**. The .env file contains the individual user environment variables that override the variables set in the /etc/environment file. You can customize your environment variables as desired by modifying your .env file.

The following example is a typical .env file:

```
export  myid=`id | sed -n -e 's/).*$//' -e 's/^.*(//p'`
#set prompt: login & system name & path
if [ $myid = root ]
        then    typeset -x PSCH='#:\${PWD}> '
                PS1="#:\${PWD}> "
        else    typeset -x PSCH='>'
                PS1="$LOGNAME@$UNAME:\${PWD}> "
                PS2=">"
                PS3="#?"
fi
export PS1 PS2 PS3
#setup my command aliases
alias   ls="/bin/ls -CF" \
        d="/bin/ls -Fal | pg" \
        rm="/bin/rm -i" \
        up="cd .."
```

Note: When modifying the .env file, ensure that newly created environment variables do not conflict with standard variables such as *MAIL*, *PS1*, *PS2*, and *IFS*.

AIXwindows startup files

Different computer systems have different ways of starting the X Server and AIXwindows.

Because different computer systems have different ways of starting the X Server and AIXwindows, consult with your system administrator to learn how to get started. Usually, the X Server and AIXwindows are started from a shell script that runs automatically when you log in. You might, however, find that you need to start the X Server or AIXwindows, or both.

If you log in and find that your display is functioning as a single terminal with no windows displayed, you can start the X Server by typing the following:

```
xinit
```

Note: Before entering this command, make sure that the pointer rests within a window that has a system prompt.

If this command does not start the X Server, check with your system administrator to ensure that your search path contains the X11 directory containing executable programs. The appropriate path might differ from one system to another.

If you log in and find one or more windows without frames, you can start AIXwindows Window Manager by typing the following:

```
mwm &
```

Because AIXwindows permits customization both by programmers writing AIXwindows applications and by users, you might find that mouse buttons or other functions do not operate as you might expect from reading this documentation. You can reset your AIXwindows environment to the default behavior by pressing and holding the following four keys:

Alt-Ctrl-Shift-!

You can return to the customized behavior by pressing this key sequence again. If your system does not permit this combination of keystrokes, you can also restore default behavior from the default root menu.

.xinitrc file
The **xinit** command uses a customizable shell script file that lists the X Client programs to start. The `.xinitrc` file in your home directory controls the windows and applications that start when you start AIXwindows.

The **xinit** command works with shell scripts in the following order:

1. The **xinit** command first looks for the *$XINITRC* environment variable to start AIXwindows.
2. If the *$XINITRC* environment variable is not found, the **xinit** command looks for the `$HOME/.xinitrc` shell script.
3. If the `$HOME/.xinitrc` shell script is not found, the **xinit** command starts the `/usr/lib/X11/$LANG/xinitrc` shell script.
4. If `/usr/lib/X11/$LANG/xinitrc` is not found, it looks for the `/usr/lpp/X11/defaults /$LANG/xinitrc` shell script. If that script is not found, it searches for the `/usr/lpp/X11/defaults/xinitrc` shell script.
5. The `xinitrc` shell script starts commands, such as the **mwm** (AIXwindows Window Manager), **aixterm**, and **xclock** commands.

The **xinit** command performs the following operations:

- Starts an X Server on the current display
- Sets up the *$DISPLAY* environment variable
- Runs the `xinitrc` file to start the X Client programs

The following example shows the part of the `xinitrc` file you can customize:

```
# This script is invoked by /usr/lpp/X11/bin/xinit

.
.
.
#***************************************************************
#  Start the X clients. Change the following lines to         *
#  whatever command(s) you desire!                           *
#  The default clients are an analog clock (xclock), an lft  *
#  terminal emulator (aixterm), and the Motif Window  Manager  *
#  (mwm).  *
```

```
#*************************************************************
exec mwm
```

.Xdefaults file

If you work in an AIXwindows interface, you can customize this interface with the `.Xdefaults` file. AIXwindows allows you to specify your preferences for visual characteristics, such as colors and fonts.

Many aspects of Windows operating system based application's appearance and behavior are controlled by sets of variables called *resources*. The visual or behavioral aspect of a resource is determined by its assigned value. There are several different types of values for resources. For example, resources that control color can be assigned predefined values such as *DarkSlateBlue* or *Black*. Resources that specify dimensions are assigned numeric values. Some resources take Boolean values (*True* or *False*).

If you do not have a `.Xdefaults` file in your home directory, you can create one with any text editor. After you have this file in your home directory, you can set resource values in it as you wish. A sample default file called `Xdefaults.tmpl` is in the `/usr/lpp/X11/defaults` directory.

The following example shows part of a typical `.Xdefaults` file:

```
*AutoRaise: on
*DeIconifyWarp: on
*warp:on
*TitleFont:andysans12
*scrollBar: true
*font: Rom10.500
Mwm*menu*foreground: black
Mwm*menu*background: CornflowerBlue
Mwm*menu*RootMenu*foreground: black
Mwm*menu*RootMenu*background: CornflowerBlue
Mwm*icon*foreground: grey25
Mwm*icon*background: LightGray
Mwm*foreground: black
Mwm*background: LightSkyBlue
Mwm*bottomShadowColor: Blue1
Mwm*topShadowColor: CornflowerBlue
Mwm*activeForeground: white
Mwm*activeBackground: Blue1
Mwm*activeBottomShadowColor: black
Mwm*activeTopShadowColor: LightSkyBlue
Mwm*border: black
Mwm*highlight:white
```

```
aixterm.foreground: green
aixterm.background: black
aixterm.fullcursor: true
aixterm.ScrollKey: on
aixterm.autoRaise: true
aixterm.autoRaiseDelay: 2
aixterm.boldFont:Rom10.500
aixterm.geometry: 80x25
aixterm.iconFont: Rom8.500
aixterm.iconStartup: false
aixterm.jumpScroll: true
aixterm.reverseWrap: true
aixterm.saveLines: 500
aixterm.scrollInput: true
aixterm.scrollKey: false
aixterm.title: AIX
```

.mwmrc file

Most of the features that you want to customize can be set with resources in your `.Xdefaults` file. However, key bindings, mouse button bindings, and menu definitions for your window manager are specified in the supplementary `.mwmrc` file, which is referenced by resources in the `.Xdefaults` file.

If you do not have a `.mwmrc` file in your home directory, you can copy it as follows:

```
cp /usr/lib/X11/system.mwmrc .mwmrc
```

Because the `.mwmrc` file overrides the system-wide effects of the `system.mwmrc` file, your specifications do not interfere with the specifications of other users.

The following example shows part of a typical `system.mwmrc` file:

```
#  DEFAULT mwm RESOURCE DESCRIPTION FILE (system.mwmrc)
#
# menu pane descriptions
#
# Root Menu Description
```

```
Menu RootMenu
{ "Root Menu"        f.title
  no-label           f.separator
  "New Window"       f.exec "aixterm &"
  "Shuffle Up"       f.circle_up
  "Shuffle Down"     f.circle_down
  "Refresh"          f.refresh
  no-label           f.separator
  "Restart"          f.restart
  "Quit"             f.quit_mwm
}
```

```
# Default Window Menu Description

Menu DefaultWindowMenu MwmWindowMenu
{ "Restore"   _R   Alt<Key>F5              f.normalize
  "Move"      _M   Alt<Key>F7              f.move
  "Size"      _S   Alt<Key>F8              f.resize
  "Minimize"  _n   Alt<Key>F9              f.minimize
  "Maximize"  _x   Alt<Key>F10             f.maximize
  "Lower"     _L   Alt<Key>F3              f.lower
  no-label                                 f.separator
  "Close"     _C   Alt<Key>F4              f.kill
}
```

```
# no acclerator window menu
Menu NoAccWindowMenu
{
  "Restore"   _R     f.normalize
  "Move"      _M     f.move
  "Size"      _S     f.resize
  "Minimize"  _n     f.minimize
  "Maximize"  _x     f.maximize
  "Lower"     _L     f.lower
  no-label           f.separator
  "Close"     _C     f.kill
}
```

```
Keys DefaultKeyBindings
{
  Shift<Key>Escape             icon|window        f.post_wmenu
  Meta<Key>space               icon|window        f.post_wmenu
  Meta<Key>Tab                 root|icon|window   f.next_key
  Meta Shift<Key>Tab           root|icon|window   f.prev_key
  Meta<Key>Escape              root|icon|window   f.next_key
  Meta Shift<Key>Escape        root|icon|window   f.prev_key
  Meta Ctrl Shift<Key>exclam   root|icon|window   f.set_behavior
}
```

```
#
# button binding descriptions
#
```

```
Buttons DefaultButtonBindings
{
  <Btn1Down>         frame|icon             f.raise
  <Btn3Down>         frame|icon             f.post_wmenu
  <Btn1Down>         root                   f.menu  RootMenu
  <Btn3Down>         root                   f.menu  RootMenu
  Meta<Btn1Down>     icon|window            f.lower
  Meta<Btn2Down>     window|icon            f.resize
  Meta<Btn3Down>     window                 f.move
}
```

```
Buttons PointerButtonBindings
{
```

```
  <Btn1Down>        frame|icon             f.raise
  <Btn2Down>        frame|icon             f.post_wmenu
  <Btn3Down>        frame|icon             f.lower
  <Btn1Down>        root                   f.menu  RootMenu
  Meta<Btn2Down>    window|icon            f.resize
  Meta<Btn3Down>    window|icon            f.move
}
```

```
#
#  END OF mwm RESOURCE DESCRIPTION FILE
#
```

Exporting shell variables (export shell command)

A *local* shell variable is a variable known only to the shell that created it. If you start a new shell, the old shell's variables are unknown to it. If you want the new shells that you open to use the variables from an old shell, export the variables to make them *global*.

You can use the **export** command to make local variables global. To make your local shell variables global automatically, export them in your `.profile` file.

Note: Variables can be exported down to child shells but not exported up to parent shells.

See the following examples:

- To make the local shell variable *PATH* global, type the following:

```
export PATH
```

- To list all your exported variables, type the following:

```
export
```

  The system displays information similar to the following:

```
DISPLAY=unix:0
EDITOR=vi
ENV=$HOME/.env
HISTFILE=/u/denise/.history
HISTSIZE=500
HOME=/u/denise
LANG=En_US
LOGNAME=denise
MAIL=/usr/mail/denise
MAILCHECK=0
MAILMSG=**YOU HAVE NEW MAIL.
USE THE mail COMMAND TO SEE YOUR MAILPATH=/usr/mail/denise?denise has mail !!!
MAILRECORD=/u/denise/.Outmail
PATH=/usr/ucb:/usr/lpp/X11/bin:/bin:/usr/bin:/etc:/u/denise:/u/denise/bin:/u/bin1
PWD=/u/denise
SHELL=/bin/ksh
```

Changing the default font (chfont command)

To change the default font at system startup, use the **chfont** or **smit** command. A *font palette* is a file that the system uses to define and identify the fonts it has available.

Note: To run the **chfont** command, you must have root authority.

chfont command

See the following examples on how to use the **chfont** command:

- To change the active font to the fifth font in the font palette, type the following:

```
chfont -a5
```

- To change the font to an italic, roman, and bold face of the same size, type the following:

```
chfont -n /usr/lpp/fonts/It114.snf /usr/lpp/fonts/Bld14.snf  /usr/lpp/fonts/Rom14.snf
```

smit command

The **chfont** command can also be run using **smit**.

To select the active font, type the following:

```
smit chfont
```

To select the font palette, type the following:

```
smit chfontpl
```

Changing control keys (stty command)

To change the keys that your terminal uses for control keys, use the **stty** command.

Your changes to control keys remain in effect until you log out. To make your changes permanent, place them in your `.profile` file.

See the following examples:

- To assign Ctrl-Z as the interrupt key, type the following:

  ```
  stty intr ^Z
  ```

  Be sure to place a space character between `intr` and ^Z.
- To reset all control keys to their default values, type the following:

  ```
  stty sane
  ```
- To display your current settings, type the following:

  ```
  stty -a
  ```

Changing your system prompt

You can change your system prompt.

Your shell uses the following prompt variables:

Item	Description
PS1	Prompt used as the normal system prompt
PS2	Prompt used when the shell expects more input
PS3	Prompt used when you have root authority

You can change any of your prompt characters by changing the value of its shell variable. Your prompt changes remain in effect until you log out. To make your changes permanent, place them in your `.env` file.

See the following examples:

- To display the current value of the *PS1* variable, type the following:

  ```
  echo "prompt is $PS1"
  ```

  The system displays information similar to the following:

  ```
  prompt is $
  ```
- To change your prompt to Ready>, type the following:

  ```
  PS1="Ready> "
  ```
- To change your continuation prompt to `Enter more->`, type the following:

  ```
  PS2="Enter more->"
  ```

- To change your root prompt to `Root->`, type the following:

```
PS3="Root-> "
```

BSD systems reference

This appendix is for system administrators who are familiar with 4.3 BSD UNIX or System V operating systems. This information explains the differences and the similarities between those systems and AIX.

Topics discussed in this appendix are:

BSD concepts

Before you start working with Berkeley Software Distribution (BSD) you need to understand some of the difference between BSD and AIX.

Introduction to AIX for BSD system managers

The following are tips to help Berkeley Software Distribution (BSD) system managers get started managing AIX.

- Start by logging in as root at the graphics console.
- Perform system management from the system console until you become experienced with the system. It is easier to work from the system console than a remote terminal. Once you are experienced with the system, you can work remotely from an xterm or an ASCII terminal.
- Take advantage of the several AIX facilities for system management tasks. They include:
  - System Management Interface Tool (SMIT). SMIT provides an interface between system managers and configuration and management commands. SMIT can help system managers perform most system administration tasks.
  - The Object Data Manager (ODM). The ODM provides routines that access objects from the ODM databases. The ODM databases contain device configuration information
  - The System Resource Controller (SRC). The SRC provides access and control of daemons and other system resources through a single interface.

Related concepts

System Resource Controller

The System Resource Controller (SRC) provides a set of commands and subroutines to make it easier for the system manager and programmer to create and control subsystems.

Related information

Configuration of a large number of devices

Major differences between 4.3 BSD and AIX

The following is a summary of the major differences between AIX and 4.3 BSD systems.

On AIX, the network daemons are started from the `/etc/rc.tcpip` file, not the `/etc/rc.local` file. The `/etc/rc.tcpip` shell script is invoked from the `/etc/inittab` file, not the `/etc/rc` file.

If the System Resource Controller (SRC) is running, the TCP/IP daemons run under SRC control. If you do not want the TCP/IP daemons running under SRC control, use the **`smit setbootup_option`** fast path to change the system to BSD-style **rc** configuration.

These network management functions available on 4.3 BSD are supported by AIX:

- Kernel-level SYSLOG logging facilities
- Access rights for UNIX domain sockets.

Configuration data storage

4.3 BSD usually stores configuration data in ASCII files. Related pieces of information are kept on the same line and record processing (sorting and searching) can be done on the ASCII file itself.

Records can vary in length and are terminated by a line feed. 4.3 BSD provides tools to convert some potentially large ASCII files to a database (dbm) format. Relevant library functions search the pair of dbm files if they exist, but search the original ASCII file if the dbm files are not found.

Some configuration data for AIX is stored in ASCII files, but often in a *stanza* format. A stanza is a set of related pieces of information stored in a group of several lines. Each piece of information has a label to make the contents of the file more understandable.

AIX also supports dbm versions of password and user information. Furthermore, the `/etc/passwd`, `/etc/group`, and `/etc/inittab` files are examples of files for AIX where the information is stored in traditional form rather than in stanza form.

Other configuration data for AIX are stored in files maintained by the Object Data Manager (ODM). System Management Interface Tool (SMIT) can manipulate and display information in ODM files. Alternatively, you can use the ODM commands directly to view these files. To query the ODM files, use the following commands:

- **odmget**
- **odmshow**.

The following ODM commands alter ODM files:

- **odmadd**
- **odmcreate**
- **odmdrop**
- **odmchange**
- **odmdelete**.

Attention: Altering ODM files incorrectly can cause the system to fail, and might prevent you from successfully restarting the system. Only use ODM commands directly on ODM files when task-specific commands, such as those generated by SMIT, is unsuccessful.

Configuration management

When a system running AIX starts up, a set of configuration-specific commands are invoked by the Configuration Manager. These configuration-specific commands are called *methods*. Methods identify the devices on the system and update the appropriate ODM files in the `/etc/objrepos` directory.

Device special files in the `/dev` directly are not preinstalled. Some special files, such as those for hard disks, are created automatically during the startup configuration process. Other special files, such as those for ASCII terminals, must be created by the system administrator by using the SMIT **Devices** menu. This information is retained in the ODM for later use by the system.

Disk management

In AIX, disk drives are referred to as *physical volumes*. Partitions are referred to as *logical volumes*. As in 4.3 BSD, a single physical volume can have multiple logical volumes. However, unlike 4.3 BSD, a single logical volume in AIX can span multiple physical volumes. To do this, you must make several physical volumes into a *volume group* and create logical volumes on the volume group.

Commands in AIX used for file system and volume management include:

- **crfs**
- **varyonvg**
- **varyoffvg**
- **lsvg**
- **importvg**
- **exportvg**.

The following 4.3 BSD commands are also available:

- **mkfs**

- **fsck**
- **fsdb**
- **mount**
- **umount**.

Differences between these commands for 4.3 BSD and for AIX are discussed in "File systems for BSD 4.3 system managers" on page 347.

4.3 BSD maintains a list of file systems in the /etc/fstab file. AIX maintains a stanza for each file system in the /etc/filesystems file.

The tn3270 command

The **tn3270** command is a link to the **telnet** command, but it uses the /etc/map3270 file and the current *TERM* environment variable value to provide 3270 keyboard mappings. Thus, the **tn3270** command operates exactly like the BSD version.

If you want to change the escape sequences from the defaults used by the **tn3270**, **telnet**, or **tn** commands, set the *TNESC* environment variable before starting these commands.

New commands

To handle new configuration and disk management systems, AIX has about 150 commands that are new to 4.3 BSD administrators.

Startup

AIX supports automatic identification and configuration of devices. Consequently, the startup process is very different from 4.3 BSD systems. In addition to the kernel, an image of a boot file system and the previous base device configuration information is loaded to a RAM disk. In the first phase of startup, sufficient configuration information is loaded and checked to permit accessing logical volumes. The paging space device is identified to the kernel and the hard disk root file system is checked. At this time, the operating system changes the root file system from the RAM disk to the hard disk and completes the startup procedure, including configuring other devices.

User authorization

4.3 BSD, and versions of AT&T UNIX operating systems before SVR4, store all user authentication information, including encrypted passwords, in the /etc/passwd file. Traditionally, the /etc/passwd file could be read by all.

On SVR4 systems, encrypted passwords are removed from the /etc/passwd file and stored in the /etc/shadow file. Only users with root authority and trusted programs (such as the /bin/login program) can read the /etc/shadow file.

AIX stores encrypted passwords in the /etc/security/passwd file. Other files in the /etc/security directory are the user and limits files. These three files define the way a user is allowed to access the system (such as using the **rlogin** or **telnet** commands) and the user's resource limits (such as file size and address space).

Printing

Most 4.3 BSD printing commands are supported with minor differences. One difference is that the /etc/qconfig file is the configuration file in AIX.

The line printing system for AIX can interoperate with the 4.3 BSD line printing system, both for submitting print jobs to 4.3 BSD systems and for printing jobs submitted from a 4.3 BSD system.

Shells

AIX supports the Bourne shell, C shell and Korn shell. The full path name for the Bourne shell program is /bin/bsh. The /bin/sh file is a hard link to the /bin/ksh file. This file can be changed by the administrator.

AIX does not support **setuid** or **setgid** for shell scripts in any shell.

Note:

1. AIX has no shell scripts that rely on the /bin/sh. However, many shell scripts from other systems rely on /bin/sh being the Bourne shell.
2. Although the Bourne shell and Korn shell are similar, the Korn shell is not a perfect superset of the Bourne shell.

Related reference

Commands for system administration for BSD 4.3 system managers
This list contains commands that are specifically for administering the environment for AIX.

File comparison table for 4.3 BSD, SVR4, and AIX

The following table compares file names and functions between 4.3 BSD, SVR4, and AIX.

Table 63. File Comparison Table

4.3 BSD File	SVR4 File	File for AIX	Database	Type (odm/ dbm)
L-Devices	Devices	Devices	no	
L-dialcodes	Dialcodes	Dialcodes	no	
L.cmds	Permissions	Permissions	no	
L.sys	Systems	Systems	no	
USERFILE	Permissions	Permissions	no	
aliases	mail/ namefiles	aliases	aliasesDB/DB	dbm
fstab	vfstab	filesystems	no	
ftpusers	ftpusers	ftpusers	no	
gettytab		N/A		
group	group	group	no	
hosts	hosts	hosts	no	
hosts.equiv	hosts.equiv	hosts.equiv	no	
inetd.conf	inetd.conf	inetd.conf	no	
map3270	N/A	map3270	no	
motd	motd	motd	no	
mtab	mnttab	N/A	no	
named.boot	named.boot	named.boot	no	
named.ca		named.ca	no	
named.hosts		named.data (See note)	no	
named.local		named.local	no	
named.pid	named.pid	named.pid	no	
named.rev		named.rev	no	
networks	networks	networks	no	
passwd	passwd	passwd	no	
printcap	qconfig	qconfig		
protocols		protocols	no	

Table 63. File Comparison Table (continued)				
4.3 BSD File	**SVR4 File**	**File for AIX**	**Database**	**Type (odm/ dbm)**
remote	remote	remote	no	
resolv.conf	resolv.conf	resolv.conf	no	
sendmail.cf	sendmail.cf	sendmail.cf	sendmail.cfDB	neither
services		services	no	
shells	shells	N/A		
stab		N/A		
syslog.conf		syslog.conf	no	
syslog.pid		syslog.pid	no	
termcap	terminfo	terminfo		
ttys	ttys	N/A	yes	odm
types		N/A		
utmp	utmp	utmp		
vfont		N/A		
vgrindefs		vgrindefs		
wtmp	wtmp	wtmp		

Note: The file names named.ca, named.hosts, named.local, and named.rev are user definable in the named.boot file. However, these are the names used for these files in the documentation for AIX.

Name and address resolution

The gethostbyname and gethostbyaddr subroutines in the libc library provide support for Domain Name Service, Network Information Services (NIS, formerly called Yellow Pages), and the /etc/hosts database.

If the /etc/resolv.conf file exists, the name server is always checked first. If the name is not resolved and NIS is running, NIS is checked. If NIS is not running, the /etc/hosts file is checked.

Online documentation and man command for BSD 4.3 system managers

AIX supports the **man-k**, **apropos**, and **whatis** commands, but the database used by these commands must first be created with the **catman-w** command.

The **man** command first searches for flat text pages in the /usr/man/cat? files. Next, it searches **nroff**-formatted pages in /usr/man/man? files. New man pages can be added in flat text or **nroff** form.

Note:

- The **man** command text pages are not provided with the system. The **catman** command creates the database from these text pages. These pages can be either flat text pages stored in the /usr/man/cat? files or **nroff**-formatted pages stored the in /usr/man/man? files.
- The Text Formatting licensed program must be installed for the **nroff** command to be available for the **man** command to read **nroff**-formatted man pages.

For more information about these commands, see **man**, **apropos**, **whatis**, and **catman**.

NFS and NIS (formerly Yellow Pages) for BSD 4.3 system managers

The following describes NFS and NIS for BSD 4.3 system managers.

Network File System (NFS) and Network Information Services (NIS) daemons are started from the `/etc/rc.nfs` file. However, before the NFS and NIS daemons can be started, the **portmap** daemon must be started in the `/etc/rc.tcpip` file. By default, the `/etc/rc.nfs` file is not invoked by the `/etc/inittab` file. If you add a line in the `/etc/inittab` file to invoke the `/etc/rc.nfs` script, it should be invoked after the `/etc/rc.tcpip` script.

If NIS is active, include a root entry prior to the `+::` (plus sign, colon, colon) entry in the `/etc/passwd` file and a system entry prior to the `+::` entry in the `/etc/group` file. This allows a system administrator to log in as root and make changes if the system is unable to communicate with the NIS server.

NFS can be configured by using the SMIT fast path, **smit nfs**. The and SMIT menus refer to NIS (formerly Yellow Pages) as NIS. Many of the NFS and NIS commands are found in the `/etc` and `/usr/etc` directory.

Some NFS environments use an **arch** command to identify machine families and types of machines. For example if you are using the IBM RS/6000, specify the `power` identifier for family (CPU), and the `ibm6000` identifier for type (machine).

User passwords for BSD 4.3 system managers

When you use the **/bin/passwd** command for AIX as the root user, you are prompted for the current root user password.

An example of using the **/bin/passwd** command follows:

```
# passwd cslater
Changing password for "cslater"
Enter root's Password or
cslater's Old password:
cslater's New password:
Re-enter cslater's
new password:
#
```

The 4.3 BSD version does not prompt for the current root user password. An example of the 4.3 BSD version follows:

```
# passwd cslater
New password:
Retype new password:
#
```

Administering BSD

There are multiple commands for BSD you can use to measure performance, print, and manage your system.

Accounting for BSD 4.3 system managers

The accounting files in the `/usr/lib/acct` directory and the system activity reporting tools in the `/usr/lib/sa` directory for AIX are identical to those available with AT&T System V Release 4 (SVR4) with the addition of 4.3 BSD accounting utilities.

Many of the accounting commands are in the `/usr/lib/acct` directory. To begin system accounting, use the `/usr/lib/acct/startup` command. If accounting is not started, commands such as **lastcomm(1)** cannot return information.

AIX provides these 4.3 BSD accounting facilities:

Item	Description
last(1)	Indicates last logins of users and terminals
lastcomm(1)	Shows in reverse order the last commands executed

Item	Description
acct(3)	Enables and disables process accounting
ac(8)	Login accounting
accton(8)	Turns system accounting on or off
sa(8)	Generally maintains system accounting files.

AIX also provides these System V Interface Definition (SVID) Issue II accounting commands and library functions:

Item	Description
acctcms(1)	Produces command usage summaries from accounting records
acctcom(1)	Displays selected process-accounting record summaries
acctcon1(1)	Converts login/logoff records to session records
acctcon2(1)	Converts login/logoff records to total accounting records
acctdisk(1)	Generates total accounting records from **diskusg(1)** command output
acctmerg(1)	Merges total accounting files into an intermediary file
accton(1)	Turns on accounting
acctprc1(1)	Processes accounting information from **acct(3)** command
acctprc2(1)	Processes output of **acctprc1(1)** command into total accounting records
acctwtmp(1)	Manipulates connect-time accounting records
chargefee(1)	Charges to login name
ckpacct(1)	Checks size of `/usr/adm/pacct` file
diskusg(1)	Generates disk accounting information
dodisk(1)	Performs disk accounting
fwtmp(1)	Converts binary records (`wtmp` file) to formatted ASCII. **Note:** The `wtmp` file is in the `/var/adm` directory.
lastlogin(1)	Updates last date on which each person logged in
monacct(1)	Creates monthly summary files
prctmp(1)	Prints session record file produced by **acctcon1(1)** command
prdaily(1)	Formats a report of yesterday's accounting information
prtacct(1)	Formats and prints any total accounting file
runacct(1)	Runs daily accounting
shutacct(1)	Called by system shutdown to stop accounting and log the reason
startup(1)	Called by system initialization to start accounting
turnacct(1)	Turns process accounting on or off
wtmpfix(1)	Corrects time/date stamps in a file using `wtmp` format

Backup for BSD 4.3 system managers

BSD 4.3 system managers can back up data.

The **tar** and **cpio** commands can move data between systems. The **tar** command for AIX is not fully compatible with the 4.3 BSD **tar** command. The **tar** command for AIX requires the **-B** flag (blocking input) if it is reading from a pipe. The AT&T **cpio** command is compatible with this version.

AIX can read and write in **dump** and **restore** command format. For example, the **backup** command for AIX with the syntax:

```
backup -0uf Device Filesystemname
```

is the same as the 4.3 BSD **dump** command with the syntax:

```
dump 0uf Device Filesystemname
```

Similarly, the **restore** command for AIX with the syntax:

```
restore -mivf Device
```

is the same as the 4.3 BSD **restore** command with the syntax:

```
restore ivf Device
```

AIX also has the 4.3 BSD **rdump** and **rrestore** commands. The only difference in the two versions is that for AIX, each argument must be preceded by a **-** (dash). For example, the following command:

```
rdump -0 -f orca:/dev/rmt0 /dev/hd2
```

is equivalent to the 4.3 BSD command:

```
rdump 0f orca:/dev/rmt0 /dev/hd2
```

The **backup** command for AIX with the following syntax:

```
backup -0f /dev/rmt0 /dev/hd2
```

is equivalent to the 4.3 BSD **dump** command with this syntax:

```
dump 0f /dev/rmt0 /dev/hd2
```

Non-IBM SCSI tape support

AIX does not directly support non-IBM SCSI tape drives. However, you can add your own header and interface that use the IBM SCSI driver.

Related concepts

System backup

Once your system is in use, your next consideration should be to back up the file systems, directories, and files. If you back up your file systems, you can restore files or file systems in the event of a hard disk crash. There are different methods for backing up information.

Related information

Adding an Unsupported Device to the System

Startup for BSD 4.3 system managers

The following discusses AIX system startup for BSD 4.3 system managers.

On 4.3 BSD systems, the **init** program is the last step in the startup procedure. The main role of the **init** program is to create processes for each available terminal port. The available terminal ports are found by reading the `/etc/ttys` file.

On System V, the **init** program is started at system initialization. The **init** process starts processes according to entries in the `/etc/inittab` file.

AIX follows the System V initialization procedure. You can edit the /etc/inittab file by directly editing the file, using the **telinit** command, or by using the following commands:

Item	Description
chitab(1)	Changes records in the /etc/inittab file
lsitab(1)	Lists records in the /etc/inittab file
mkitab(1)	Makes records in the /etc/inittab file
rmitab(1)	Removes records in the /etc/inittab file

Changes made to the /etc/inittab file take effect the next time the system is rebooted, or when the **telinit q** command is run.

Finding and examining files for BSD 4.3 system managers

The following is a list of the BSD file commands that AIX supports.

AIX supports the following 4.3 BSD file commands:

- **which**
- **whereis**
- **what**
- **file**.

AIX does not support the 4.3 BSD **fast find** syntax of the **find** command. At this time, there is no replacement function. The following **ffind** shell script can be used to simulate the functionality:

```
#!/bin/bsh
PATH=/bin
for dir in /bin /etc /lib /usr
do
find $dir -print | egrep $1
done
```

The syntax for the **ffind** script is:

```
ffind Filename
```

Paging space for BSD 4.3 system managers

The following commands assist in managing paging space (also known as swap space).

Item	Description
chps(1)	Changes attributes of a paging space
lsps(1)	List attributes of a paging space
mkps(1)	Add an additional paging space to the system
rmps(1)	Removes a paging space from the system
swapoff(1)	Deactivates one or more paging spaces
swapon(1)	Specifies additional devices for paging and swapping

If a large paging space is required, place one paging logical volume for each hard disk. This allows scheduling of paging across multiple disk drives.

Changing the default startup to permit 4.3 BSD ASCII configuration

You can administer network interfaces for AIX through the SMIT and ODM files, or through 4.3 BSD ASCII configuration files.

To administer network interfaces through 4.3 BSD ASCII configuration files, uncomment the commands in the `/etc/rc.net` file below the heading:

```
# Part II - Traditional
Configuration
```

Then if you want flat file configuration and SRC support, edit the `/etc/rc.net` file and uncomment the **hostname**, **ifconfig**, and **route** commands with the appropriate parameters.

If you want flat file configuration without SRC support, use the **smit setbootup_option** fast path to change the system to BSD-style **rc** configuration. This option configures the system to use the `/etc/rc.bsdnet` file at startup. You also have to edit the `/etc/rc.bsdnet` file and uncomment the **hostname**, **ifconfig**, and **route** commands with the appropriate parameters.

Additional options for ifconfig and netstat commands

The following is a list of additional options for the **ifconfig** and **netstat** commands.

The **ifconfig** command for AIX has the following additional options:

mtu
: The *mtu* variable specifies the maximum transmission unit (MTU) used on the local network (and local subnets) and the MTU used for remote networks. To maximize compatibility with Ethernet and other networks, set both the Token-Ring and Ethernet default *mtu* value to 1500.

allcast
: The **allcast** flag sets the Token-Ring broadcast strategy. Setting the **allcast** flag optimizes connectivity through Token-Ring bridges. Clearing the **allcast** flag (by specifying `-allcast`) minimizes excess traffic on the ring.

The **netstat** command for AIX has the **-v** flag. The **netstat -v** command prints driver statistics such as transmit byte count, transmit error count, receive byte count, and receive error count. For more information about the **ifconfig** and **netstat** commands, see **ifconfig** and **netstat**.

Additional network management commands

The following additional commands are supported on AIX.

Item	Description
securetcpip	The **securetcpip** shell script enables controlled access mode, which provides enhanced network security. It disallows the running of several unsecured TCP/IP programs, such as the `tftp`, `rcp`, `rlogin`, and `rsh` programs. It also restricts the use of the `.netrc` file.
gated	The **gated** command provides MIB support for SNMP.

Item	Description
no	The **no** command sets network options that include: **dogticks** Sets timer granularity for `ifwatchdog` routines **subnetsarelocal** Determines if packet address is on the local network **ipsendredirects** Specifies whether the kernel should send redirect signals **ipforwarding** Specifies whether the kernel should forward packets **tcp_ttl** Specifies the time-to-live for Transmission Control Protocol (TCP) packets **udp_ttl** Specifies the time-to-live for User Datagram Protocol (UDP) packets **maxttl** Specifies the time-to-live for Routing Information Protocol (RIP) packets **ipfragttl** Specifies the time-to-live for Internet Protocol (IP) fragments **lowclust** Specifies a low water mark for cluster **mbuf** pool **lowmbuf** Specifies a low water mark for the **mbuf** pool **thewall** Specifies the maximum amount of memory that is allocated to the `mbuf` and cluster `mbuf` pool **arpt_killc** Specifies the time in minutes before an inactive complete Address Resolution Protocol (ARP) entry is deleted
iptrace	The **iptrace** command provides interface-level packet tracing for Internet protocols.

ipreport	The **ipreport** command formats the trace into human-readable form. An example of using this command is the following:
	`iptrace -i en0 /tmp/iptrace.log`
	`# kill iptrace daemon`
	`kill `ps ax | grep iptrace | awk '{ print $1 }'``
	`ipreport /tmp/iptrace.log | more`

Importing a BSD 4.3 password file

You can import a BSD 4.3 password file into AIX.

To import a BSD 4.3 password file, perform the following steps:

1. Copy the BSD 4.3 password file to the `/etc/passwd` file and enter:

   ```
   pwdck -y ALL
   ```

2. Update the `/etc/security/limits` file with a null stanza for any new users.

   The **usrck** command does this, but using the **usrck** command can cause problems unless the `/etc/group` file is imported with the `/etc/passwd` file. For more information about the **usrck** command, see **usrck**.

Attention: If the `/etc/security/limits` file is modified, the stack must not exceed 65,536 bytes. If it does, running the **usrck** command can cause problems. Change the stack size to 65,536 and run **usrck** command again.

3. Run the **grpck** and **usrck** commands to verify group and user attributes.

Editing the password file for BSD 4.3 system managers

The following explains how to change entries in the password file and how to administer passwords on AIX in a BSD 4.3 manner.

In AIX, the **lsuser**, **mkuser**, **chuser**, and **rmuser** commands are provided for managing passwords. All of these commands can be used by running SMIT. However, all of these commands deal with only one user at a time.

For more information about these commands, see **lsuser**, **mkuser**, **chuser**, and **rmuser**.

Note: Using an editor to change several user name entries at one time requires editing of several files simultaneously, because passwords are stored in the `/etc/security/passwd` file, authorization information is stored in the `/etc/security/user` file, and the remaining user data is stored in the `/etc/passwd` file.

AIX does not support the **vipw** command but does support the **mkpasswd** command. However, you can still administer passwords in AIX in a BSD 4.3 manner. Use the following procedure:

1. Put a BSD 4.3 password file in the `/etc/shadow` file.
2. Change the permissions to the file by entering:

```
chmod 000 /etc/shadow
```

3. Place the following **vipw** shell script in the `/etc` directory:

```
------------------------------------------------------
----
#!/bin/bsh
#
# vipw. Uses pwdck for now. May use usrck someday
#
PATH=/bin:/usr/bin:/etc:/usr/ucb # Add to this if your editor is
                                 # some place else
if [ -f /etc/ptmp ] ; then
                 echo "/etc/ptmp exists. Is someone else using vipw?"
        exit 1
fi
if [ ! -f /`which "$EDITOR" | awk '{ print $1 }'` ] ; then
        EDITOR=vi
fi
cp /etc/shadow /etc/ptmp
if (cmp /etc/shadow /etc/ptmp) ; then
        $EDITOR /etc/ptmp
else
        echo cannot copy shadow to ptmp
        exit 1
fi
if (egrep "^root:" /etc/ptmp >/dev/null) ; then
        cp /etc/ptmp /etc/shadow ; cp /etc/ptmp /etc/passwd
        chmod 000 /etc/passwd /etc/shadow
        pwdck -y ALL 2>1 >/dev/null # return code 114 may change
                rc=$?
        if [ $rc -eq 114 ]; then
                chmod 644 /etc/passwd
                        rm -f /etc/passwd.dir /etc/passwd.pag
                mkpasswd /etc/passwd
                        # update /etc/security/limits, or ftp
                        # will fail
                else
                         pwdck -y ALL
                fi
else
        echo bad entry for root in ptmp
fi
rm /etc/ptmp
------------------------------------------------------------
```

4. If you use the **vipw** shell script or the **mkpasswd** command, be aware that SMIT, and the **mkuser**, **chuser**, and **rmuser** commands do not use the **mkpasswd** command. You must run:

```
mkpasswd /etc/passwd
```

   to update the /etc/passwd.dir and /etc/passwd.pag files.

   Attention: Initialization of the *IFS* variable and the trap statements guard against some of the common methods used to exploit security holes inherent in the **setuid** feature. However, the **vipw** and **passwd** shell scripts are intended for relatively open environments where compatibility is an important consideration. If you want a more secure environment, use only the standard commands for AIX.

5. Put the following **passwd** shell script in the /usr/ucb directory:

```
------------------------------------------------------
#!/bin/ksh
#
# matches changes to /etc/security/passwd file with changes to
#/etc/shadow
#
IFS=" "
PATH=/bin
trap "exit 2" 1 2 3 4 5 6 7 8 10 12 13 14 15 16 17 18 21 22 \
        23 24 25 27 28 29 30 31 32 33 34 35 36 60 61 62
if [ -n "$1" ]; then
        USERNAME=$1
else
        USERNAME=$LOGNAME
fi
if [ -f /etc/ptmp ]; then
        echo password file busy
        exit 1
fi
        trap "rm /etc/ptmp; exit 3" 1 2 3 4 5 6 7 8 10 12 13 \
                14 15 16 17 18 21 22 23 24 25 27 28 29 30 31 \
                32 33 34 35 36 60 61 62
if (cp /etc/security/passwd /etc/ptmp) ; then
        chmod 000 /etc/ptmp else
        rm -f /etc/ptmp exit 1
fi
if ( /bin/passwd $USERNAME ) ; then
        PW=` awk ' BEGIN { RS = "" }
                $1 == user { print $4 } ' user="$USERNAME:" \
/etc/security/passwd `
else
        rm -f /etc/ptmp
        exit 1
fi
rm -f /etc/ptmp
awk -F: '$1 == user { print $1":"pw":"$3 ":"$4":"$5":"$6":"$7 }
        $1 != user { print $0 }' user="$USERNAME" pw="$PW" \
                /etc/shadow > /etc/ptmp
chmod 000 /etc/ptmp
mv -f /etc/ptmp /etc/shadow
------------------------------------------------------------
```

6. Change the permissions to the **passwd** script by entering:

```
chmod 4711 /usr/ucb/passwd
```

7. Ensure that each user *PATH* environmental variable specifies that the /usr/ucb directory be searched before the /bin directory.

Performance measurement and tuning for BSD 4.3 system managers

Thie following discusses AIX device attributes and performance measurement and tuning.

All devices on AIX have attributes associated with them. To view device attributes, enter:

```
lsattr -E -l devicename
```

Any attributes with the value True can be modified with the command:

```
chdev -l devicename -a attr=value
```

Attention: Changing device parameters incorrectly can damage your system.

By default, the maximum number of processes per user is 40. The default value might be too low for users who have many windows open simultaneously. The following command can be used to change the value systemwide:

```
hdev -l sys0 -a maxuproc=100
```

This example changes the maximum number to 100. The new value is set once the system has restarted.

To view the current setting of this and other system attributes, type:

```
lsattr -E -l sys0
```

The `maxmbuf` attribute is not currently supported by the mbuf services.

AIX supports the **vmstat** and **iostat** commands, but not the **systat** command or load averages. For more information about these commands, see **vmstat** and **iostat**.

Printers for BSD 4.3 system managers

The AIX operating system supports two printing subsystems: 4.3 BSD and System V.

The System V style of printing subsystem uses System V Release 4 commands, queues, and files and is administered the same way. The following paragraphs describe what you need to know to manage the 4.3 BSD style of printing subsystem. You control which subsystem is made active through SMIT. Only one subsystem can be active at a time.

Printing is managed by programs and configurations in the `/usr/lpd` directory. The design, configuration, queueing mechanism, and daemon processes of the 4.3 BSD and printer subsystems for AIX are different. However, they both use the `lpd` protocol for remote printing. Both systems use `/etc/hosts.lpd`, if it exists, or `/etc/host.equiv` otherwise. The printer subsystem for AIX offers a gateway to 4.3 BSD printer subsystems, so systems using AIX can submit print jobs to 4.3 BSD systems and accept print jobs submitted by 4.3 BSD systems.

The `/etc/printcap` file of 4.3 BSD does not exist in AIX. This file is a combination of spooler configuration and printer capability database. Users need to understand the format and keywords of the `printcap` file to set up a printer correctly.

The `/etc/qconfig` file of AIX contains only the spooler configuration information. The printer capability is defined in the ODM predefined or customized database. You can use the **mkvirprt** (make virtual printer) command to define to the system the capabilities of a particular printer.

To make printer lp0 available to print on the remote host viking, put the following in a 4.3 BSD system `/etc/printcap` file:

```
lp0|Print on remote printer attached to
viking:Z
:lp=:rm=viking:rp=lp:st=/usr/spool/lp0d
```

To do the same in AIX, put the following in the `/etc/qconfig` file:

```
lp0:
        device = dlp0
        host = viking
        rq = lp
dlp0:
        backend = /usr/lib/lpd/rembak
```

AIX supports the following printer commands and library functions:

Item	Description
cancel(1)	Cancels requests to a line printer
chquedev(1)	Changes the printer or plotter queue device names
chvirprt(1)	Changes the attribute values of a virtual printer
disable(1)	Disables a printer queue
enable(1)	Enables a printer queue
hplj(1)	Postprocesses **troff** output for HP LaserJetII with the K cartridge
ibm3812(1)	Postprocesses **troff** output for IBM 3812 Mod 2 Pageprinter
ibm3816(1)	Postprocesses **troff** output for IBM 3816 Pageprinter
ibm5587G(1)	Postprocesses **troff** output for IBM 5587G with 32x32/24x24 cartridge
lp(1)	Sends requests to a line printer
lpr(1)	Enqueues print jobs
lprm(1)	Removes jobs from the line printer spooling queue
lpstat(1)	Displays line printer status information
lptest(1)	Generates the line printer ripple pattern
lsallqdev(1)	Lists all configured printer queue device names within a queue
lsvirprt(1)	Displays the attribute values of a virtual printer
mkque(1)	Adds a printer queue to the system
mkquedev(1)	Adds a printer queue device to the system
mkvirprt(1)	Makes a virtual printer
pac(1)	Prepares printer/plotter accounting records
piobe(1)	Print Job Manager for the printer backend
pioburst(1)	Generates burst pages (header and trailer pages) for printer output
piocmdout(3)	Subroutine that outputs an attribute string for a printer formatter
piodigest(1)	Digests attribute values for a virtual printer definition and stores
pioexit(3)	Subroutine that exits from a printer formatter
pioformat(1)	Drives a printer formatter
piofquote(1)	Converts certain control characters destined for PostScript printers
piogetstr(3)	Subroutine that retrieves an attribute string for a printer formatter
piogetvals(3)	Subroutine that initializes Printer Attribute database variables for printer formatter
piomsgout(3)	Subroutine that sends a message from a printer formatter
pioout(1)	Printer backend's device driver interface program
piopredef(1)	Creates a predefined printer data stream definition
proff(1)	Formats text for printers with personal printer data streams
qadm(1)	Performs system administration for the printer spooling system
qconfig(4)	Configures a printer queueing system
qstatus(1)	Provides printer status for the print spooling system
restore(3)	Restores the printer to its default state

Item	Description
rmque(1)	Removes a printer queue from the system
rmquedev(1)	Removes a printer or plotter queue device from the system
rmvirprt(1)	Removes a virtual printer
splp(1)	Changes or displays printer driver settings
xpr(1)	Formats a window dump file for output to a printer

Related information
Printer Overview for System Management

Commands for system administration for BSD 4.3 system managers

This list contains commands that are specifically for administering the environment for AIX.

Item	Description
bosboot(1)	Initializes a boot device.
bootlist(1)	Alters the list of boot devices (or the ordering of these devices in the list) available to the system.
cfgmgr(1)	Configures devices by running the programs in `/etc/methods` directory.
chcons(1)	Redirects the system console to device or file, effective next startup
chdev(1)	Changes the characteristics of a device
chdisp(1)	Changes the display used by the low-function terminal (LFT) subsystem.
checkcw(1)	Prepares constant-width text for the **troff** command.
checkeq(1)	Checks documents formatted with memorandum macros.
checkmm(1)	Checks documents formatted with memorandum macros.
checknr(1)	Checks `nroff` and `troff` files.
chfont(1)	Changes the default font selected at boot time.
chfs(1)	Changes attributes of a file system.
chgroup(1)	Changes attributes for groups.
chgrpmem(1)	Changes the administrators or members of a group.
chhwkbd(1)	Changes the low function terminal (LFT) keyboard attributes stored in the Object Data Manager (ODM) database.
chitab(1)	Changes records in the `/etc/inittab` file.
chkbd(1)	Changes the default keyboard map used by the low-function terminal (LFT) at system startup.
chkey(1)	Changes your encryption key.
chlang	Sets *LANG* environment variable in the `/etc/environment` file for the next login.
chlicense(1)	There are two types of user licensing, fixed and floating. Fixed licensing is always enabled, and the number of licenses can be changed through the **-u** flag. Floating licensing can be enabled or disabled (on or off) through the **-f** flag
chlv(1)	Changes the characteristics of a logical volume
chnamsv(1)	Changes TCP/IP-based name service configuration on a host

Item	Description
chprtsv(1)	Changes a print service configuration on a client or server machine
chps(1)	Changes attributes of a paging space.
chpv(1)	Changes the characteristics of a physical volume in a volume group.
chque(1)	Changes the queue name.
chquedev(1)	Changes the printer or plotter queue device names.
chssys(1)	Changes a subsystem definition in the subsystem object class.
chtcb(1)	Changes or queries the trusted computing base attribute of a file.
chtz	Changes the system time zone information.
chuser(1)	Changes attributes for the specified user.
chvfs(1)	Changes entries in the `/etc/vfs` file.
chvg(1)	Sets the characteristics of a volume group.
chvirprt(1)	Changes the attribute values of a virtual printer.
crfs(1)	Adds a file system.
crvfs(1)	Creates entries in the `/etc/vfs` file.
exportvg(1)	Exports the definition of a volume group from a set of physical volumes.
extendvg(1)	Adds physical volumes to a volume group.
grpck(1)	Verifies the correctness of a group definition.
importvg(1)	Imports a new volume group definition from a set of physical volumes.
lsallq(1)	Lists the names of all configured queues.
lsallqdev(1)	Lists all configured printer and plotter queue device names within a specified queue.
lsdisp(1)	Lists the displays currently available on the system.
lsfont(1)	Lists the fonts available for use by the display.
lsfs(1)	Displays the characteristics of file systems.
lsgroup(1)	Displays the attributes of groups.
lsitab(1)	Lists the records in the `/etc/inittab` file.
lskbd(1)	Lists the keyboard maps currently available to the low-function terminal (LFT) subsystem.
lslicense(1)	Displays the number of fixed licenses and the status of floating licensing.
lslpp(1)	Lists optional program products.
lsnamsv(1)	Shows name service information stored in the database.
lsprtsv(1)	Shows print service information stored in the database.
lsps	Lists paging space and attributes.
lsque(1)	Displays the queue stanza name.
lsquedev(1)	Displays the device stanza name.
lssrc(1)	Gets the status of a subsystem, a group of subsystems, or a subserver.
lsuser(1)	Displays attributes of user accounts.
lsvfs(1)	Lists entries in the `/etc/vfs` file.

Item	Description
mkcatdefs(1)	Preprocesses a message source file.
runcat(1)	Pipes the output data from the **mkcatdefs** command to the `gencat` command.
mkdev(1)	Adds a device to the system.
mkfont(1)	Adds the font code associated with a display to the system.
mkfontdir(1)	Creates a `fonts.dir` file from a directory of font files.
mkgroup(1)	Creates a new group.
mkitab(1)	Makes records in the `/etc/inittab` file.
mklv(1)	Creates a logical volume.
mklvcopy(1)	Adds copies to a logical volume.
mknamsv(1)	Configures TCP/IP-based name service on a host for a client.
mknotify(1)	Adds a notify method definition to the notify object class.
mkprtsv(1)	Configures TCP/IP-based print service on a host.
mkps(1)	Add an additional paging space to the system.
mkque(1)	Adds a printer queue to the system.
mkquedev(1)	Adds a printer queue device to the system.
mkserver(1)	Adds a subserver definition to the subserver object class.
mkssys(1)	Adds a subsystem definition to the subsystem object class.
mksysb	Backs up mounted file systems in the rootvg volume group for subsequent reinstallation.
mkszfile	Records size of mounted file systems in the rootvg volume group for reinstallation.
mktcpip(1)	Sets the required values for starting TCP/IP on a host.
mkuser(1)	Creates a new user account.
mkuser.sys(1)	Customizes a new user account.
mkvg(1)	Creates a volume group.
mkvirprt(1)	Makes a virtual printer.
odmadd(1)	Adds objects to created object classes.
odmchange(1)	Changes the contents of a selected object in the specified object class.
odmcreate(1)	Produces the `.c` (source) and `.h` (include) files necessary for ODM application development and creates empty object classes.
odmdelete(1)	Deletes selected objects from a specified object class.
odmdrop(1)	Removes an object class.
odmget(1)	Retrieves objects from the specified object classes and places them into an `odmadd` input file.
odmshow(1)	Displays an object class definition on the screen.
pwdck(1)	Verifies the correctness of local authentication information.
redefinevg	Redefines the set of physical volumes of the given volume group in the device configuration database.

Item	Description
reducevg(1)	Removes physical volumes from a volume group. When all physical volumes are removed from the volume group, the volume group is deleted.
reorgvg(1)	Reorganizes the physical partition allocation for a volume group.
restbase(1)	Restores customized information from the boot image.
rmdel(1)	Removes a delta from a Source Code Control System (SCCS) file.
rmdev(1)	Removes a device from the system.
rmf(1)	Removes folders and the messages they contain.
rmfs(1)	Removes a file system.
rmgroup(1)	Removes a group.
rmitab(1)	Removes records in the `/etc/inittab` file.
rmlv(1)	Removes logical volumes from a volume group.
rmlvcopy(1)	Removes copies from a logical volume.
rmm(1)	Removes messages.
rmnamsv(1)	Unconfigures TCP/IP-based name service on a host.
rmnotify(1)	Removes a notify method definition from the notify object class.
rmprtsv(1)	Unconfigures a print service on a client or server machine.
rmps(1)	Removes a paging space from the system.
rmque(1)	Removes a printer queue from the system.
rmquedev(1)	Removes a printer or plotter queue device from the system.
rmserver(1)	Removes a subserver definition from the subserver object class.
rmssys(1)	Removes a subsystem definition from the subsystem object class.
rmuser(1)	Removes a user account.
rmvfs(1)	Removes entries in the `/etc/vfs` file.
rmvirprt(1)	Removes a virtual printer.
savebase(1)	Saves base customized device data in the ODM onto the boot device.
swapoff(1)	Deactivates one or more paging space.
swapon(1)	Specifies additional devices for paging and swapping.
syncvg(1)	Synchronizes logical volume copies that are not current.
usrck(1)	Verifies the correctness of a user definition.
varyoffvg(1)	Deactivates a volume group.
varyonvg(1)	Activates a volume group.

Related concepts

Major differences between 4.3 BSD and AIX
The following is a summary of the major differences between AIX and 4.3 BSD systems.

Cron for BSD 4.3 system managers

The **cron** daemon for this operating system is similar to the System V Release 2 **cron** daemon.

An entry in the `/etc/inittab` file starts the **cron** daemon.

Devices for BSD 4.3 system managers

The following discusses devices for BSD 4.3 system managers.

A device on a 4.3 BSD system is accessible to an application only when:

- The device is physically installed and functioning.
- The driver for the device is in the kernel.
- The device special files for the device exist in the /dev directory.

A device on AIX is accessible to an application only when:

- The device is physically installed and functioning.
- The driver for the device is in the kernel or in a loaded kernel extension.
- The device special files for the device exist in the /dev directory.
- The object database in the /etc/objrepos directory contains entries for the device that match the physical configuration.

The device specific programs called *methods*, found in the /etc/methods directory, maintain the object database. The methods are invoked by the Configuration Manager (accessed through the **cfgmgr** command) and other commands.

If a device can no longer be accessed by an application program, it can mean that the hardware is faulty or it can mean that the configuration database in the /etc/objrepos directory is damaged.

The **cfgmgr** command processes the configuration database in the **/etc/objrepos** directory and is processed at startup time by the **cfgmgr** command (the Configuration Manager).

The pseudocode below shows the Configuration Manager logic:

```
/* Main */
While there are rules in the Config_Rules database
        {
        Get the next rule and execute it
        Capture stdout from the last execution
        Parse_Output(stdout)
        }
/* Parse Output Routine */
/* stdout will contain a list of devices found */
Parse_OutPut(stdout)
        {
        While there are devices left in the list
                {
                Lookup the device in the database
                if (!defined)
                        Get define method from database and execute
                if (! configured)
                        {
                        Get config method from database and execute
                        Parse_Output(stdout)
                         }
                }
        }
```

UUCP for BSD 4.3 system managers

The following table lists the UUCP commands and files.

Item	Description
Dialers(4)	Lists modems used for BNU remote communications links
Maxuuxqts(4)	Limits the number of instances of the BNU **uuxqt** daemons that can run
Permissions(4)	Specifies BNU command permissions for remote systems
Poll(4)	Specifies when the BNU program should poll remote systems
Systems(4)	Lists remote computers with which the local system can communicate

Item	Description
rmail(1)	Handles remote mail received through BNU
uucheck(1)	Checks for files and directories required by BNU
uuclean(1)	Removes files from the BNU spool directory
uucleanup(1)	Deletes selected files from the BNU spooling directory
uucpadm(1)	Enters basic BNU configuration information
uudemon.admin(1)	Provides periodic information on the status of BNU file transfers
uudemon.cleanu(1)	Cleans up BNU spooling directories and log files
uudemon.hour(1)	Initiates file transport calls to remote systems using the BNU program
uudemon.poll(1)	Polls the systems listed in the BNU Poll file
uulog(1)	Provides information about BNU file-transfer activities on a system
uupoll(1)	Forces a poll of a remote BNU system
uuq(1)	Displays the BNU job queue and deletes specified jobs from the queue
uusnap(1)	Displays the status of BNU contacts with remote systems
uustat(1)	Reports the status of and provides limited control over BNU operations

AIX also provides the 4.3 BSD **uuencode** and **uudecode** commands. The HDB **uugetty** command is not supported. For information about these commands, see **uuencode** and **uudecode**.

Related information
BNU file and directory structure

File systems for BSD 4.3 system managers

Similar commands are used to mount and unmount file systems.

AIX uses the `/etc/filesystem` file to list file system device information, and has similar commands for mounting and unmounting file systems.

/etc/filesystems file and /etc/fstab file

4.3 BSD systems store lists of block devices and mount points in the `/etc/fstab` file. SVR4 systems store block devices and mount point information in `/etc/vfstab` file. AIX stores block device and mount points information in `/etc/filesystems` file.

The **crfs**, **chfs**, and **rmfs** commands update the `/etc/filesystems` file.

4.3 BSD system administrators might be interested in the *check* variable in the `/etc/filesystems` file. The *check* variable can be set to the value `True`, `False`, or to a number. For example, you can specify `check=2` in the `/etc/filesystems` file. The number specifies the pass of the **fsck** command that will check this file system. The *check* parameter corresponds to the fifth field in an `/etc/fstab` file record.

There is no dump frequency parameter in the `/etc/filesystems` file.

File system support on AIX

AIX supports various file systems.

AIX supports disk quotas.

AIX does not allow mounting of diskettes as file systems.

The syntax of the **mount** and **umount** commands for AIX differs from 4.3 BSD and from SVR4 versions of these commands. The commands to mount and unmount all file systems at once are shown for all three systems in the following table:

mount and **unmount** Commands			
Function	**Syntax for this operating system**	**4.3 BSD Syntax**	**SVR4 Syntax**
mount all file systems	**mount all**	**mount -a**	**mountall**
unmount all file systems	**umount all**	**umount -a**	**umountall**

See ../devicemanagement/file_sys.html#file_sys for more information.

Terminals for BSD 4.3 system managers

The following discusses terminals for BSD 4.3 system managers.

Traditionally, 4.3 BSD system managers enable or disable terminal ports by modifying the `/etc/ttys` file and sending a HUP signal to the **init** program.

AIX stores terminal port information in the ODM and starts terminals when the **init** program reads the `/etc/inittab` file. In AIX, use the SMIT interface to configure terminal ports.

There is no fixed mapping between the port and the device special file name in the `/dev` directory. Consequently, it is confusing to system managers who are new to AIX which port is to be configured. When using SMIT, the first planar serial port (physically labeled **s1**) is referred to as location **00-00-S1**, adapter **sa0**, and port **s1** in the SMIT menus. The second planar serial port (physically labeled **s2**) is referred to as location **00-00-S2**, adapter **sa1**, and port **s2**.

Use the **penable** and **pdisable** commands to enable and disable a port.

termcap and terminfo

Like System V, this operating system uses `terminfo` entries in `/usr/lib/terminfo/?/*` files.

Users with 4.3 BSD Systems might find the following commands helpful:

captoinfo(1)
Converts a `termcap` file to a `terminfo` file

tic(1)
Translates the `terminfo` files from source to compiled format.

This operating system includes source for many `terminfo` entries. Some of these might need to be compiled with the **tic** command. The `termcap` file is provided in `/lib/libtermcap/termcap.src` file.

Input and output redirection

The AIX operating system allows you to manipulate the input and output (I/O) of data to and from your system by using specific I/O commands and symbols.

You can control input by specifying the location from which to gather data. For example, you can specify to read input while data is entered on the keyboard (standard input) or to read input from a file. You can control output by specifying where to display or store data. You can specify to write output data to the screen (standard output) or to write it to a file.

Because AIX is a multitasking operating system, it is designed to handle processes in combination with each other.

Related concepts

Commands for displaying file contents (pg, more, page, and cat commands)
The **pg**, **more**, and **page** commands allow you to view the contents of a file and control the speed at which your files are displayed.

Input and output redirection in the Korn shell or POSIX shell

Before the Korn shell executes a command, it scans the command line for redirection characters. These special notations direct the shell to redirect input and output.

Standard input, standard output, and standard error files

When a command begins running, it usually expects that the following files are already open: standard input, standard output, and standard error (sometimes called *error output* or *diagnostic* output).

A number, called a *file descriptor,* is associated with each of these files, as follows:

Item	Description
File descriptor 0	Standard input
File descriptor 1	Standard output
File descriptor 2	Standard error (diagnostic) output

A child process normally inherits these files from its parent. All three files are initially assigned to the workstation (0 to the keyboard, 1 and 2 to the display). The shell permits them to be redirected elsewhere before control is passed to a command.

When you enter a command, if no file name is given, your keyboard is the *standard input*, sometimes denoted as *stdin*. When a command finishes, the results are displayed on your screen.

Your screen is the *standard output*, sometimes denoted as *stdout*. By default, commands take input from the standard input and send the results to standard output.

Error messages are directed to standard error, sometimes denoted as *stderr*. By default, this is your screen.

These default actions of input and output can be varied. You can use a file as input and write results of a command to a file. This is called *input/output redirection*.

The output from a command, which normally goes to the display device, can be redirected to a file instead. This is known as *output redirection*. This is useful when you have a lot of output that is difficult to read on the screen or when you want to put files together to create a larger file.

Though not used as much as output redirection, the input for a command, which normally comes from the keyboard, can also be redirected from a file. This is known as *input redirection*. Redirection of input lets you prepare a file in advance and then have the command read the file.

Standard output redirection

When the notation >*filename* is added to the end of a command, the output of the command is written to the specified file name. The > symbol is known as the *output redirection operator*.

Any command that outputs its results to the screen can have its output redirected to a file.

Redirecting output to a file

The output of a process can be redirected to a file by typing the command followed by the output redirection operator and file name.

For example, to redirect the results of the **who** command to a file named `users`, type the following:

```
who > users
```

Note: If the `users` file already exists, it is deleted and replaced, unless the **noclobber** option of the **set** built-in **ksh** (Korn shell) or **csh** (C shell) command is specified.

To see the contents of the `users` file, type the following:

```
cat users
```

A list similar to the following is displayed:

```
denise    lft/0 May 13 08:05
marta     pts/1 May 13 08:10
endrica   pts/2 May 13 09:33
```

Redirecting output to append to a file

When the notation >> *filename* is added to the end of a command, the output of the command is appended to the specified file name, rather than writing over any existing data. The >> symbol is known as the *append redirection operator*.

For example, to append `file2` to `file1`, type the following:

```
cat file2 >> file1
```

Note: If the `file1` file does not exist, it is created, unless the **noclobber** option of the **set** built-in **ksh** (Korn shell) or **csh** (C shell) command is specified.

Creating a text file with redirection from the keyboard

Used alone, the **cat** command uses whatever you type at the keyboard as input. You can redirect this input to a file.

Press Ctrl-D on a new line to signal the end of the text.

At the system prompt, type the following:

```
cat > filename
This is a test.
^D
```

Text file concatenation

You can combine multiple files into one file. Combining various files into one file is known as *concatenation*.

The following example creates `file4`, which consists of `file1`, `file2`, and `file3`, appended in the order below.

See the following examples:

- At the system prompt, type the following:

  ```
  cat file1 file2 file3 > file4
  ```

- The following example shows a common error when concatenating files:

  ```
  cat file1 file2 file3 > file1
  ```

  Attention: In this example, you might expect the **cat** command to append the contents of `file1`, `file2`, and `file3` into `file1`. The **cat** command creates the output file first, so it actually erases the contents of `file1` and then appends `file2` and `file3` to it.

Standard input redirection

When the notation < *filename* is added to the end of a command, the input of the command is read from the specified file name. The < symbol is known as the *input redirection operator*.

Note: Only commands that normally take their input from the keyboard can have their input redirected.

For example, to send the file `letter1` as a message to user `denise` with the **mail** command, type the following:

```
mail denise < letter1
```

Discarding output with the /dev/null file

The `/dev/null` file is a special file. This file has a unique property: it is always empty. Any data sent to `/dev/null` is discarded. This is a useful feature when you run a program or command that generates output that you want to ignore.

For example, you have a program named `myprog` that accepts input from the screen and generates messages while it is running that you would rather not see on your screen. To read input from the file `myscript` and discard the standard output messages, type the following:

```
myprog < myscript >/dev/null
```

In this example, `myprog` uses the file `myscript` as input, and all standard output is discarded.

Standard error and other output redirection

In addition to the standard input and standard output, commands often produce other types of output, such as error or status messages known as diagnostic output. Like standard output, standard error output is written to the screen unless it is redirected.

To redirect standard error or other output, use a file descriptor. A *file descriptor* is a number associated with each of the I/O files that a command ordinarily uses. File descriptors can also be specified to redirect standard input and standard output. The following numbers are associated with standard input, output, and error:

Item	Description
0	Standard input (keyboard)
1	Standard output (display)
2	Standard error (display)

To redirect standard error output, type the file descriptor number 2 in front of the output or append redirection symbols (> or > >) and a file name after the symbol. For example, the following command takes the standard error output from the **cc** command where it is used to compile the `testfile.c` file and appends it to the end of the ERRORS file:

```
cc testfile.c 2>> ERRORS
```

Other types of output can also be redirected using the file descriptors from `0` through 9. For example, if the **cmd** command writes output to file descriptor 9, you can redirect that output to the `savedata` file with the following command:

```
cmd 9> savedata
```

If a command writes to more than one output, you can independently redirect each one. Suppose that a command directs its standard output to file descriptor 1, directs its standard error output to file descriptor 2, and builds a data file on file descriptor 9. The following command line redirects each of these outputs to a different file:

```
command > standard 2> error 9> data
```

Redirecting output to inline input (here) documents

You can redirect output to inline input (here) documents.

If a command is in the following form:

```
command << eofstring
```

and *eofstring* is any string that does not contain pattern-matching characters, then the shell takes the subsequent lines as the standard input of *command* until the shell reads a line consisting of only *eofstring* (possibly preceded by one or more tab characters). The lines between the first *eofstring* and the second are frequently referred to as an *inline input document*, or a *here document*. If a hyphen (-) immediately follows the << redirection characters, the shell strips leading tab characters from each line of the **here** document before it passes the line to *command*.

The shell creates a temporary file containing the **here** document and performs variable and command substitution on the contents before passing the file to the command. It performs pattern matching on file names that are part of command lines in command substitutions. To prohibit all substitutions, quote any character of the *eofstring*:

```
command << \eofstring
```

The **here** document is especially useful for a small amount of input data that is more conveniently placed in the shell procedure rather than kept in a separate file (such as editor scripts). For example, you could type the following:

```
cat <<- xyz
   This message will be shown on the
   display with leading tabs removed.
   xyz
```

Related concepts

Input and output redirection in the Korn shell or POSIX shell

Before the Korn shell executes a command, it scans the command line for redirection characters. These special notations direct the shell to redirect input and output.

Redirecting output using pipes and filters

You can connect two or more commands so that the standard output of one command is used as the standard input of another command. A set of commands connected this way is known as a *pipeline*.

The connection that joins the commands is known as a *pipe*. Pipes are useful because they let you tie many single-purpose commands into one powerful command. You can direct the output from one command to become the input for another command using a pipeline. The commands are connected by a pipe (|) symbol.

When a command takes its input from another command, modifies it, and sends its results to standard output, it is known as a *filter*. Filters can be used alone, but they are especially useful in pipelines. The most common filters are as follows:

- sort
- more
- pg

See the following examples:

- The **ls** command writes the contents of the current directory to the screen in one scrolling data stream. When more than one screen of information is presented, some data is lost from view. To control the output so the contents display screen by screen, you can use a pipeline to direct the output of the **ls** command to the **pg** command, which controls the format of output to the screen. For example, type the following:

  ```
  ls | pg
  ```

  In this example, the output of the **ls** command becomes the input for the **pg** command. Press Enter to continue to the next screen.

  Pipelines operate in one direction only (left to right). Each command in a pipeline runs as a separate process, and all processes can run at the same time. A process pauses when it has no input to read or when the pipe to the next process is full.

- Another example of using pipes is with the **grep** command. The **grep** command searches a file for lines that contain strings of a certain pattern. To display all your files created or modified in July, type the following:

```
ls -l | grep Jul
```

  In this example, the output of the **ls** command becomes the input for the **grep** command.

Displaying program output and copying to a file (tee command)

The **tee** command, used with a pipe, reads standard input, then writes the output of a program to standard output and simultaneously copies it into the specified file or files. Use the **tee** command to view your output immediately and at the same time, store it for future use.

For example, type the following:

```
ps -ef | tee program.ps
```

This displays the standard output of the **ps -ef** command on the display device, and at the same time, saves a copy of it in the `program.ps` file. If the `program.ps` file already exists, it is deleted and replaced unless the **noclobber** option of the **set** built-in command is specified.

For example, to view and save the output from a command to an existing file:

```
ls -l | tee -a program.ls
```

This displays the standard output of **ls -l** at the display device and at the same time appends a copy of it to the end of the `program.ls` file.

The system displays information similar to the following, and the `program.ls` file contains the same information:

```
-rw-rw-rw-   1 jones   staff   2301   Sep 19    08:53 161414
-rw-rw-rw-   1 jones   staff   6317   Aug 31    13:17 def.rpt
-rw-rw-rw-   1 jones   staff   5550   Sep 10    14:13 try.doc
```

Clearing your screen (clear command)

Use the **clear** command to empty the screen of messages and keyboard input.

At the prompt, type the following:

```
clear
```

The system clears the screen and displays the prompt.

Sending a message to standard output

Use the **echo** command to display messages on the screen.

For example, to write a message to standard output, at the prompt, type the following:

```
echo Please insert diskette . . .
```

The following message is displayed:

```
Please insert diskette . . .
```

For example, to use the **echo** command with pattern-matching characters, at the prompt, type the following:

```
echo The back-up files are: *.bak
```

The system displays the message The `back-up files are:` followed by the file names in the current directory ending with `.bak`.

Appending a single line of text to a file (echo command)

Use the **echo** command, used with the append redirection operator, to add a single line of text to a file.

For example, at the prompt, type the following:

```
echo Remember to back up mail files by end of week.>>notes
```

This adds the message `Remember to back up mail files by end of week.` to the end of the file `notes`.

Copying your screen to a file (capture and script commands)

Use the **capture** command, which emulates a VT100 terminal, to copy everything printed on your terminal to a file that you specify. Use the **script** command to copy everything printed on your terminal to a file that you specify, without emulating a VT100 terminal.

Both commands are useful for printing records of terminal dialogs.

For example, to capture the screen of a terminal while emulating a VT100, at the prompt, type the following:

```
capture screen.01
```

The system displays information similar to the following:

```
Capture command is started. The file is screen.01.
Use ^P to dump screen to file screen.01.
You are now emulating a vt100 terminal.
Press Any Key to continue.
```

After entering data and dumping the screen contents, stop the **capture** command by pressing Ctrl-D or typing `exit` and pressing Enter. The system displays information similar to the following:

```
Capture command is complete. The file is screen.01.
You are NO LONGER emulating a vt100 terminal.
```

Use the **cat** command to display the contents of your file.

For example, to capture the screen of a terminal, at the prompt, type the following:

```
script
```

The system displays information similar to the following:

```
Script command is started. The file is typescript.
```

Everything displayed on the screen is now copied to the `typescript` file.

To stop the **script** command, press Ctrl-D or type `exit` and press Enter. The system displays information similar to the following:

```
Script command is complete. The file is typescript.
```

Use the **cat** command to display the contents of your file.

Command to display text in large letters on your screen (banner command)

The **banner** command displays ASCII characters to your screen in large letters.

Each line in the output can be up to 10 digits (or uppercase or lowercase characters) in length.

For example, at the prompt, type the following:

```
banner GOODBYE!
```

The system displays GOODBYE! in large letters on your screen.

Command summary for input and output redirection

The following are commands for input and output redirection.

Item	Description
>	“Standard output redirection” on page 349
<	“Standard input redirection” on page 350
> >	“Redirecting output to append to a file” on page 350
|	“Redirecting output using pipes and filters” on page 352
banner	Writes ASCII character strings in large letters to standard output
capture	Allows terminal screens to be dumped to a file
clear	Clears the terminal screen
echo	Writes character strings to standard output
script	Allows terminal input and output to be copied to a file
tee	Displays the standard output of a program and copies it into a file

AIX kernel recovery

Beginning with AIX 6.1, the kernel can optionally recover from errors in selected routines, avoiding an unplanned system outage.

Kernel recovery is disabled by default. If kernel recovery is enabled, the system might pause for a short time during a kernel recovery action. This time is generally less than two seconds. The following actions occur immediately after a kernel recovery action:

- The system console displays the following message:

```
---------------------------------------------------------------------------
        A kernel error recovery action has occurred. A recovery log
         has been logged in the system error log.
---------------------------------------------------------------------------
```

- AIX adds an entry into the error log. You can send the error log data to IBM for service, similar to sending data from a full system termination. The following is a sample recovery error log entry:

```
LABEL:          RECOVERY
Date/Time:       Fri Feb 16 14:04:17 CST 2007
Type:            INFO
Resource Name:   RMGR
Description
Kernel Recovery Action
Detail Data
Live Dump Base Name
RECOV_20070216200417_0000
Function Name
w_clear
FRR Name
w_init_clear_frr
Symptom String
273
EEEE00009627A072
F10001001B18BBC0
w_clear+D0
wdog0030+288
test_index+4C
Recovery Log Data
0001 0000 0000 0000 F000 0000 2FFC AEB0 0000 0111 0000 0000 0000 0000 0021 25BC
8000 0000 0002 9032 EEEE 0000 9627 A072 F100 0100 1B18 BBC0 0000 0000 0000 0000
```

```
0000 0001 0000 0000 0006 0057 D2FF 8C00 0001 0148 0500 0000 8000 0000 0002 9032
.....
```

- AIX generates a live dump. The data from a live dump is located by default in the `/var/adm/ras/livedump` directory and the file is named **RECOV_*timestamp_sequence***, where *timestamp* specifies the time of the kernel recovery occurrence, and *sequence* specifies the number of times that kernel recovery has been invoked. You can send live dump data to IBM for service, similar to sending data from a full system termination. For more information about live dumps, see live dumps in *Kernel Extensions and Device Support Programming Concepts*.

Attention: Some functions might be lost after a kernel recovery, but the operating system remains in a stable state. If necessary, shut down and restart your system to restore the lost functions.

Memory and processor considerations

AIX maintains data on the status of kernel recovery during mainline kernel operations. When kernel recovery is enabled, additional processor instructions are required to maintain the data and additional memory is required to save the data. The impact to processor usage is minimal. Additional memory consumption can be determined by the following equation, where *maxthread* is the maximum number of threads running on the system and *procnum* is the number of processors:

```
memory required = 4 KB x maxthread + 128 KB x procnum
```

As show in the following example, a system with 16 processors and a maximum of 1000 threads consumes an additional 6304 KB:

```
4 x 1000 + 128 x 16 = 6304 KB
```

Enabling and disabling kernel recovery

You can enable or disable kernel recovery from the SMIT path interface.

To enable, or disable kernel recovery, use the following SMIT path:

Problem Determination > **Kernel Recovery** > **Change Kernel Recovery State** > **Change Next Boot Kernel Recovery State**

You can also display the current kernel recovery state by using the following SMIT path:

Problem Determination > **Kernel Recovery** > **Show Kernel Recovery State**

AIX Event Infrastructure for AIX and AIX clusters-AHAFS

AIX Event Infrastructure for AIX and AIX clusters comprise an event monitoring framework for monitoring predefined and user-defined events.

Introduction to the AIX Event Infrastructure

The AIX Event Infrastructure is an event monitoring framework for monitoring predefined and user-defined events.

In the AIX Event Infrastructure, an event is defined as any change of a state or a value that can be detected by the kernel or a kernel extension at the time the change occurs. The events that can be monitored are represented as files in a pseudo file system. Some advantages of the AIX Event infrastructure are:

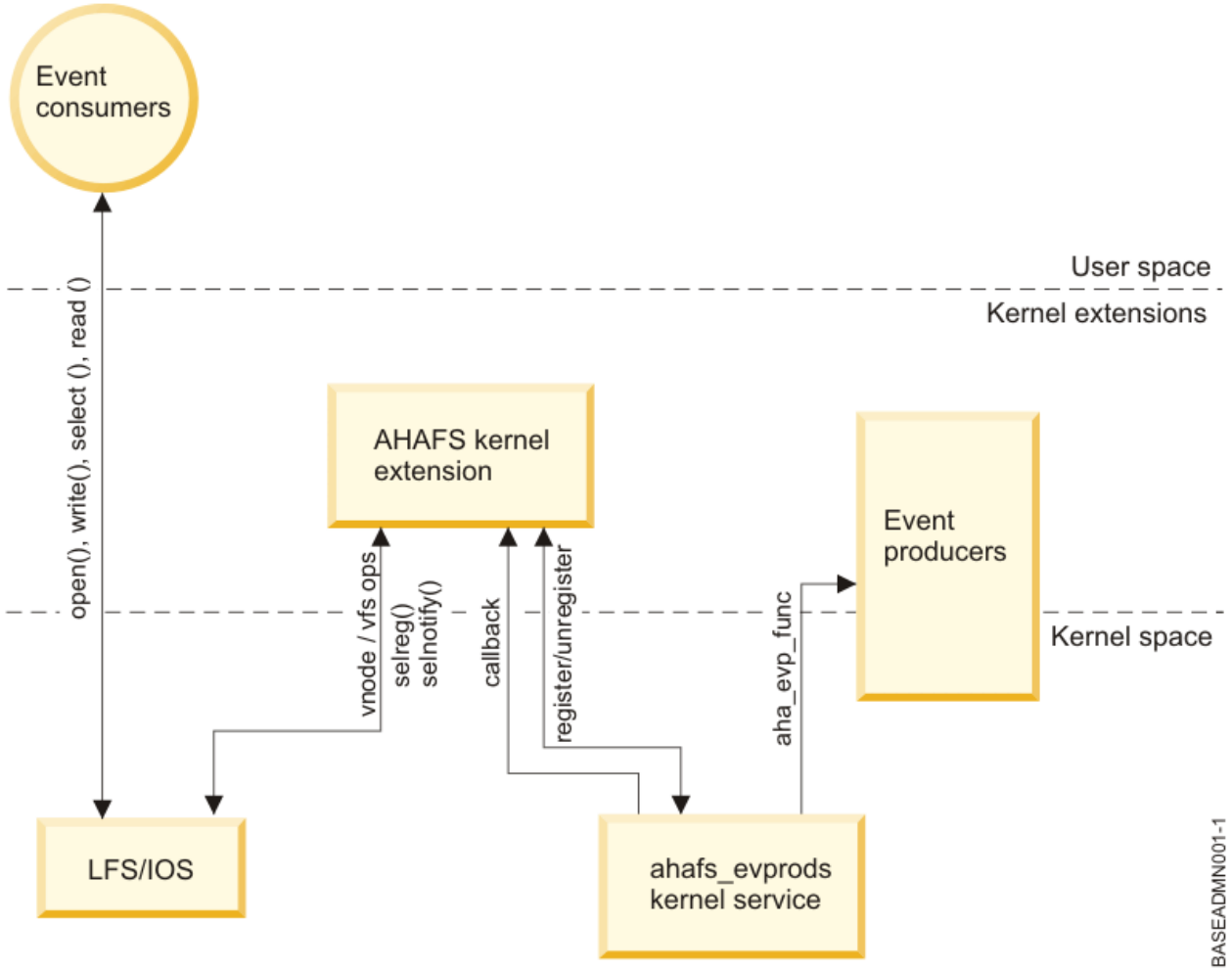
- There is no need for constant polling. Users monitoring the events are notified when those events occur.
- Detailed information about an event (such as stack trace and user and process information) is provided to the user monitoring the event.
- Existing file system interfaces are used so that there is no need for a new application programming interface (API).

- Control is handed to the AIX Event Infrastructure at the exact time the event occurs.

AIX Event Infrastructure components

The AIX Event Infrastructure is made up of the following four components:

- The kernel extension implementing the pseudo file system.
- The event consumers that consume the events.
- The event producers that produce events.
- The kernel component that serve as an interface between the kernel extension and the event producers.



AIX Event Infrastructure kernel extension

The AIX Event Infrastructure kernel extension implements the pseudo file system.

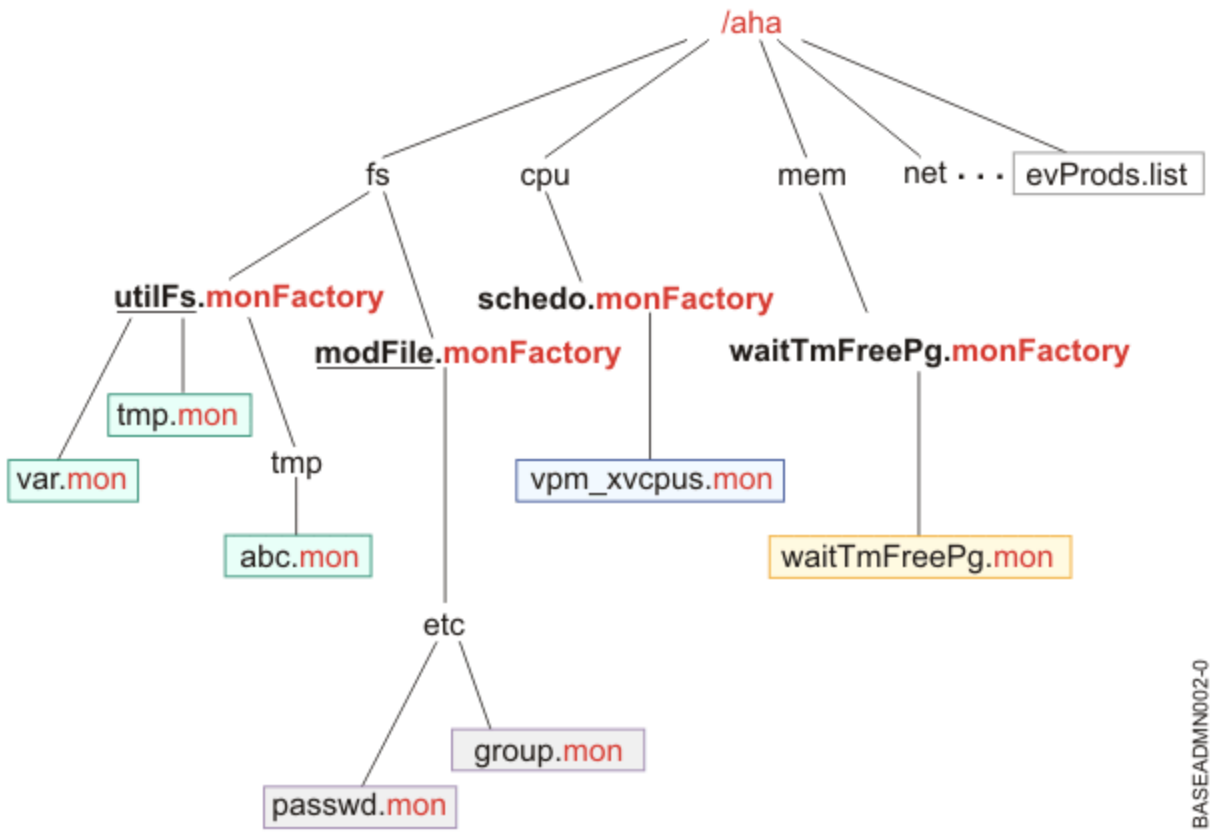
All events are represented as files in this file system. There are four file object types:

- **.list files**: There is only one **.list** file in the pseudo file system **evProds.list**. This is a special file which when read, will return the names of all currently defined event producers.
- **.monFactory directories**: Monitor factories are a special type of directory. These are directory representations of the event producers. Monitor factory directories and their parent subdirectories are automatically created for the user.
- **subdirectories**: Sub directories are used both for ease of management and to represent full path names for monitor files (see **.mon files**).

- **.mon files**: The monitor files represent the events that can be monitored. The full pathname of a monitor file from its parent monitor factory, minus the **.mon** extension is the full representation of the event being monitored. For example, the file **/aha/fs/modFile.monFactory/etc/password.mon** is used to monitor the modifications to the **/etc/passwd** file. Monitor files can only exist underneath a monitor factory.

No other regular files can be created in this pseudo file system. Since the AIX Event Infrastructure file system is an in-memory file system, there is a maximum of 32 KB of inodes that may exist. The number of inodes used will be displayed in the **df** command output.

An example of the layout of an AIX Event Infrastructure file system is shown below:



Note:

The **evProds.list** file exists directly under the root of the file system, and contains the list of event producers that are defined and usable under this operating system instance.

Using the LFS interface, the AIX Event Infrastructure will translate text input written to monitor files into specifications on how the user wants to be notified of event occurrences. Once a user has issued a **select()** or a blocking **read()** call to signify the beginning of their monitoring, the AIX Event Infrastructure will notify the corresponding event producer to start monitoring the specified event.

When an event occurrence is detected, the AIX Event Infrastructure will notify all waiting consumers whose monitoring criteria have been met.

Event consumers

Event consumers are user space processes that are waiting on events to occur.

Consumers set up event monitoring by writing information to a monitor file specifying how and when they should be notified. Consumers may wait for event notification in a **select()** call or a blocking **read()** call.

The AIX Event Infrastructure is not thread safe. Processes should not use multiple threads to monitor the same event.

Event producers

Event producers are sections of code within the kernel or a kernel extension that can detect an event.

When a monitored event occurs, the event producer notifies the AIX Event Infrastructure kernel extension and sends any associated information about the event to pass on to the consumer.

Currently, there are two main classes of event producers:

- Those that monitor for a state change
- Those that monitor for a value exceeding user-specified thresholds

ahafs_evprods kernel service

The **ahafs_evprods** kernel service facilitates communication between the AIX Event Infrastructure kernel extension and event producers.

To facilitate communication between the AIX Event Infrastructure kernel extension and event producers, the **ahafs_evprods** kernel service is exported. Inside the kernel, a list of registered event producers is used to look up event producers and to pass information between appropriate event producers and the kernel extension.

Setting up the AIX Event Infrastructure

Steps required for setting up the AIX Event Infrastructure.

The only steps necessary to set up the AIX Event Infrastructure are:

1. Install the **bos.ahafs** fileset.
2. Create the directory for the desired mount point.
3. Run the following command:

   ```
   mount –v ahafs <mount point> <mount point>
   ```

   Example

   ```
   mkdir /aha
   mount -v ahafs /aha /aha
   ```

   Mounting an AIX Event Infrastructure file system will automatically load the kernel extension and create all monitor factories. Only one instance of an AIX Event Infrastructure file system may be mounted at a time. An AIX Event Infrastructure file system may be mounted on any regular directory, but it is suggested that users use **/aha**.

High-level view of how the AIX Event Infrastructure works

A consumer may monitor multiple events, and multiple consumers may monitor the same event. Each consumer may monitor value-based events with a different threshold value. To handle this, the AIX Event Infrastructure kernel extension keeps a list of each consumer's information including:

- Specified wait type (**WAIT_IN_READ** or **WAIT_IN_SELECT**)
- Level of information requested
- Threshold (s) for which to monitor (if monitoring a threshold value event)
- A buffer used to hold information about event occurrences.

Event information is stored per-process so that different processes monitoring the same event do not alter the event data. When a consumer process reads from a monitor file, it will only read its own copy of the event data.

Typical flow of monitoring an event

The steps in monitoring an event is described in this topic.

1. A process attempts to open or create a monitor file.
2. AIX Event Infrastructure passes the pathname of the monitor file to the appropriate event producer. The event producer verifies that the monitor file represents a valid event and that the process has access to monitor the event.
3. The process writes information to the file specifying:
   a. The wait type (**WAIT_TYPE**=**WAIT_IN_READ** or **WAIT_TYPE**=**WAIT_IN_SELECT**). The default wait type is **WAIT_IN_SELECT**.
   b. When to be notified. For state change events, the user must specify **CHANGED=YES**. For threshold value events, the user may specify **THRESH_HI**=<value>, **THRESH_LO**=<value>, or both, depending on the capabilities of the associated event producer. There is no default for this specification, and **CHANGED=YES** and **THRESH_*=**<value> may not both be specified.
4. AIX Event Infrastructure will then allocate the per-process block to store this information if one does not already exist for this process and fill it with the information written by the user.
5. The process issues **select()** or a blocking **read()** on the monitor file
6. AIX Event Infrastructure will call **ahafs_evprods** to check that the thresholds specified are valid for this particular event. For example, the **utilFs** event producer does not allow values of > 100%. If the threshold is not valid, the **select()** or **read()** call will return **RC_FROM_EVPROD** and upon a read of the monitor file will be **EINVAL** returned.
7. For threshold value event producers, only one value is sent to the event producer for each threshold (**hi** or **lo**) for monitoring. At **select()** or blocking **read()** time, AIX Event Infrastructure will register this new threshold with the event producer if one of the following is true:
   a. If no other process is monitoring this event, the threshold (s) specified by this consumer are registered with the event producer.
   b. If there are other processes monitoring this event, then if the **THRESH_LO** specified by the consumer is higher than the currently monitored low threshold OR if the **THRESH_HI** specified by the consumer is lower than the currently monitored high threshold AIX Event Infrastructure calls into the **ahafs_evprods** kernel service to update the currently monitored threshold.
8. Upon return from **ahafs_evprods** kernel service, the actual value of the event is returned (in some cases). If the actual value returned has already met or exceeded either threshold, the **read()** or **select()** call will return immediately and the **RC_FROM_EVPROD** logged in the event buffer will be **EALREADY**. The **read()** or **select()** calls will return 0.
9. For state change event producers, the **ahafs_evprods** function is always called to register the event.
10. Upon a successful registration, AIX Event Infrastructure sets up notification. For consumers waiting in **select()**, the notification is set up through **selreg()**. For consumers blocking in a **read()** call, the thread is put to sleep with **e_sleep_thread()**.
11. Once an event producer detects that an event has occurred, it will notify AIX Event Infrastructure with information regarding the event (i.e. information about the process triggering the event, the current value, the return code, etc.).
12. During this callback from the event producer, AIX Event Infrastructure will:
    a. Determine the **ahaNode** corresponding to the event
    b. Search the list of waiting consumers to determine whose thresholds have been met or exceeded to notify with the **selnotify()** or **e_wakeup()** call. All consumers waiting on a state change event will be notified.
13. Once the process is notified of the event, it reads from the monitor file to obtain event data. An example of output from an event is below.

An example output for a state change event producer who has specified that a stack trace should be taken:

```
BEGIN_EVENT_INFO
TIME_tvsec=1269377315
TIME_tvnsec=955475223
SEQUENCE_NUM=0
PID=2490594
UID=0
UID_LOGIN=0
GID=0
PROG_NAME=cat
RC_FROM_EVPROD=1000
END_EVENT_INFO
```

An example for a threshold value event:

```
BEGIN_EVENT_INFO
TIME_tvsec=1269378095
TIME_tvnsec=959865951
SEQUENCE_NUM=0
CURRENT_VALUE=2
RC_FROM_EVPROD=1000
END_EVENT_INFO
```

Note: Due to the asynchronous nature of process notification, the current value returned may be stale by the time a process reads the monitor file. Users are notified when the threshold is first met or exceeded, but other operations which may alter the values being monitored will not be blocked.

Using the AIX Event Infrastructure

All directories in the AIX Event Infrastructure file system have an access mode of 1777 and all files have access mode of 0666.

Currently, all directories in the AIX Event Infrastructure file system have a mode of 1777 and all files have a mode of 0666. These modes cannot be changed, but the ownership of files and directories may be changed. Access control for monitoring events is done at the event producer level. Creation / modification times are not maintained and are always returned as the current time when issuing **stat ()** on a file object within the pseudo file system. Any attempt to modify these times will return an error.

Monitoring events

Creating the monitor file

The monitor file corresponding to the event must be created to monitor an event.

Before monitoring an event, the monitor file corresponding to the event must be created. The AIX Event Infrastructure does support **open()** with the **O_CREAT** flag. As an example, we will follow the steps required to monitor the file system **/filesys/clj-fs** for a utilization of 90%.

- The necessary subdirectories must also be created:

```
mkdir /aha/fs/utilFs.monFactory/filesys
```

- Open the file **/aha/fs/utilFs.monFactory/filesys/clj-fs.mon**.

Before a monitor file can be created, the AIX Event Infrastructure kernel extension will call the event producer to determine if the event being requested is valid and to determine if the user has sufficient authority to monitor the specified event. Here are some of the common errors which can be returned from a monitor file create or open:

Table 64. Return Codes	
Return Code	**Details**
ENODEV	There is no event corresponding to the path specified. **Note:** An **ENODEV** error may still be returned when trying to open an existing monitor file when the event no longer exists.
EPERM	User does not have permission to monitor the specified event.
ENOTSUP	The event specified does not support monitoring by AIX Event Infrastructure.

Writing to the monitor file

The consumer process writes to the monitor file to specify how and when it should be notified of events.

Once the monitor file desired is created and opened, the consumer process will write to the monitor file to specify how and when it should be notified of events. This data is written in **<key>=<value>** pairs which may be separated by a ; or whitespace. The acceptable **<key>=<value>** pairs are as follows:

Table 65. Acceptable <key>=<value> pairs		
Key	**Acceptable Values**	**Details**
CHANGED	YES	Specifies that the event to be monitored is of **AHAFS_THRESHOLD_STATE** type and that the consumer should be notified when the state of the event changes.
THRESH_HI	Unsigned, 64 bit integer, specified in decimal	This key specifies the high threshold for the event. Once the event has reached this threshold (equal to or greater), the consumer will be notified. **Note:** While this is a 64 bit integer, some event producers may have limits on what values can actually be monitored. For example, acceptable values for **THRESH_HI** for the **utilFs** event producer are between 1 and 100, inclusive. The validity of the threshold for the event producer is not checked at write time, but rather at **select()** or blocking **read()** time.

Table 65. Acceptable <key>=<value> pairs (continued)

Key	Acceptable Values	Details
THRESH_LO	Unsigned, 64 bit integer, specified in decimal	This key specifies the low threshold for the event. Once the event has reached this threshold (equal to or less than), the consumer will be notified. **Note:** While this is a 64 bit integer, some event producers may have limits on what values can actually be monitored. For example, acceptable values for **THRESH_LO** for the **utilFs** event producer are between 1 and 100, inclusive. The validity of the threshold for the event producer is not checked at write time, but rather at **select()** or blocking **read()** time.
WAIT_TYPE	WAIT_IN_SELECT (default), WAIT_IN_READ	Specifies how the consumer will wait for the event. If the consumer wishes to block for the event in a **select()** call, they should specify **WAIT_IN_SELECT**. If the consumer wishes to block for the event in a **read()** call, they should specify **WAIT_IN_READ**.

Table 65. Acceptable <key>=<value> pairs (continued)		
Key	**Acceptable Values**	**Details**
INFO_LVL	1, 2 (default), or 3	Specifies what event data should be logged in the user's buffer: • **INFO_LVL=1** will log the timestamp of the event, sequence number, event producer return code, user information*, process information*, program name*, and current value of the event (if applicable). • **INFO_LVL=2** will log all data from level 1, plus the message from the event producer, if applicable. • **INFO_LVL=3** will log all data from level 2, plus the stack of the event if applicable. **Note:** The user information, process information, program name and stack trace are only available for event producers which have specified the **AHAFS_STKTRACE_AVAILABLE** flag. Not all event producers pass messages. See the event producer documentation to determine what info is available. Examples of the event output are shown in the "Reading Event Data" on page 368 section.
NOTIFY_CNT	-1 (default), or any value between 1 and 32767, inclusive	The **NOTIFY_CNT** specifies how many times the event should occur before the process is notified. If the -1 value is specified, the consumer will be notified upon every occurrence of the event and every event occurrence will be logged in the user buffer. If the consumer specifies a positive, non zero value, the consumer will be blocked until the event has occurred the specified number of times. Once the event has occurred the specified number of times, no more events will be logged until the consumer blocks in another **select()** or blocking **read()** call. See the "Waiting on events" on page 366 section for more information.

Table 65. Acceptable <key>=<value> pairs (continued)		
Key	**Acceptable Values**	**Details**
CLUSTER	YES	If the system is part of a cluster and the cluster is active, consumers may specify this key to be notified of occurrences of this event on other nodes in the cluster. Not all event producers support cluster-wide monitoring. This feature is **off** by default. See the “Cluster events” on page 387 section for more information.
BUF_SIZE	A positive integer, up to 1048576	This key specifies the size of the buffer which should be used to record event data, specified in bytes. The default size is 2048, and the smallest size allocated will be 1024 bytes, even if the consumer requests a smaller size.

Writing information to the monitor file only prepares for a subsequent **select()** or blocking **read()** call. Monitoring does not start until a **select()** or blocking **read()** is done.

For example, to monitor the file system **/filesys/clj-fs** for the first occurrence of a utilization of 90% in a blocking **read()** call, the following string is written to the **/aha/fs/utilFs.monFactory/filesys/clj-fs.mon** file:

```
WAIT_TYPE=WAIT_IN_READ THRESH_HI=90 NOTIFY_CNT=1
```

Possible return codes from a **write()** call to a monitor file:

Table 66. Return Codes	
Return Code	**Details**
EINVAL	If an invalid value is given for any of the above keys, the write to the monitor file will fail with **EINVAL**. In addition, if the notify parameters (**CHANGED** or **THRESH_HI/LO**) specified do not match with the capabilities of the event producer, the write will fail with **EINVAL**. For example, if the consumer specifies **CHANGED=YES** for the **utilFs** event producer (which only monitors for **THRESH_HI/LO**), the write call will return **EINVAL**. Specifying **CLUSTER=YES** without an active cluster will also result in **EINVAL**.
EBUSY	If there is another thread in the process which is currently waiting on the event, a write to the monitor file by any other thread will return **EBUSY**.
ESTALE	The monitor file has been deleted. In order to monitor this event, the file descriptor will need to be closed, then reopened with **O_CREAT**

Table 66. Return Codes (continued)	
Return Code	**Details**
ENOMEM	Unable to allocate temporary memory or memory for the event buffer.
ENOSPC	A maximum of 512 processes may monitor a monitor file. If there are already 512 processes with this file open who have written to it, the write will fail with **ENOSPC**.

Waiting on events

Monitoring specifications are written to the monitor file.

Once monitoring specifications are successfully written to the monitor file, the consumer process will block for an event occurrence using **select()** or **read()**. Consumers are only notified of events that occur once they block in **select()** or **read()**. There are three ways that the process can return from **select()** or a blocking **read()**:

1. The event has occurred the specified number of times.
   - Non-error case. Consumer should read event data to determine how to handle the event.
2. There was a problem when setting up the event inside the AIX Event Infrastructure kernel extension.

   Errors may occur before the event is registered for monitoring with the event producer:
   - **read()**
     - If there is another thread waiting in read, the read will fail with **EBUSY**
     - If there was no write done before this read, the read will just return 0, with 0 bytes read.
   - **select()**

     Note:

     Due to the implementation of the select system call, in order for **select()** to return an error, the underlying file system operations must return **EBADF**. As a result, if any of the following conditions are met, **select()** will return **EBADF**.
     - Another thread is attempting a select
     - The monitor file has been deleted
     - There was no write done specifying monitoring specifications
     - There was an error when registering with the IOS subsystem

     In these cases, there will be no event data to read.
3. There was a problem setting up the event with the event producer.

   If an attempt is made to register the event with the event producer, an entry will be logged into the buffer for the consumer to read. To determine what error occurred, the **RC_FROM_EVPROD** returned in the event data should be referenced in the event producer's documentation. Note that the event output for this case will only contain the timestamp, sequence number and return code from the event producer, regardless of what **INFO_LVL** has been specified. See “Reading Event Data” on page 368 for an example.

   In this case, **select()** will return **EBADF**, but **read()** will return the return code from the underlying **uio_move** operation.

If the consumer process has specified a **NOTIFY_CNT** greater than 1, information about each event occurrence will be logged in the consumer's buffer until the number of events request have occurred. The consumer process will only be woken up if the event has occurred the requested number of times, or an

unavailable event has occurred. Once the consumer process is woken up, it will no longer be monitoring the event until it re-issues a **select()** or blocking **read()** call for the monitor file.

If a consumer has specified a **NOTIFY_CNT** of -1, the consumer process will be woken up after each occurrence of the event, and any event which occurs after the initial successful **select()** or blocking **read()** will be logged in the consumer buffer.

The **select()** and **read()** calls will not block if there is unread event data in the buffer.

Unavailable Event Occurrences

For some event producers, there may be event occurrences that cause the monitored event to become invalid.

Some examples are:

- The death of a process for **processMon** and **pidProcessMon** .
- The unmounting of a file system containing monitored files for **modDir** and **modFile** .
- The unmounting of a file system which is being monitored by **utilFs** .
- The removal or renaming of a file which is being monitored by **modDir** or **modFile**
- The removal of an event producer which is currently being used to monitor events (The **RC_FROM_EVPROD** will be **ENODEV** in this case).

Once an unavailable event occurrence has been triggered, the consumers may not continue to monitor for that event until it becomes valid again. Examples of events becoming valid again:

- The remounting of a monitored file system.
- The recreation of a monitored file that was deleted.
- The recreation of a process that was being monitored.

When a local unavailable event is triggered, the AIX Event Infrastructure kernel extension will remove the monitor file(s) affected. When a monitor file is deleted, consumers who still have the file open will be able to read their event data, but will not be able to write or block waiting for an event occurrence on that monitor file. When an unavailable event occurrence is encountered by the consumer, they should take the appropriate action (which will presumably cause the event to become valid again), close the file descriptor for the monitor file, and re-open the monitor file with the **O_CREAT** flag.

Local unavailable event occurrences will also cause **select()** and **read()** to unblock before the requested number of events occurrences have been triggered if the consumer has specified a **NOTIFY_CNT > 1**. For example, if a consumer is monitoring file **/foo** with a **NOTIFY_CNT=3**, the consumer will return from **select()** or **read()** if **/foo** is removed even if it is the first occurrence of an event with **/foo**.

Using AIX Event Infrastructure for polling

AIX Event Infrastructure does not require that event producers always maintain the current value of events which may be monitored.

This is to allow for greater performance since event producers do not have the overhead of maintaining this value if no one is monitoring for occurrences of the event.

This creates a problem when using synchronous polling. Since it is not always possible to obtain the current value of an event at every point in time, **poll()** or synchronous **select()** calls are handled in the following way:

- When a process issues **select()** or **poll()** on a monitor file for the first time, the AIX Event Infrastructure kernel extension will register that event for monitoring with the event producer.
  - For threshold value event producers who do maintain the current value, the current value will be returned to the AIX Event Infrastructure kernel extension upon event registration. This value will be checked against the consumer threshold value at this time. If the consumer's threshold has been exceeded, the **select()** or **poll()** will indicate that the event has occurred and will have an **RC_FROM_EVPROD** of **EALREADY**.

- POLLSYNC flags are ignored. An event remains registered with the event producer until the event occurs the specified number of times, or until the user closes the file.
- Subsequent **poll()** calls will have the following behavior:
  - If the event has not yet occurred, the call will return with no return events
  - If the event has occurred the specified number of times since the last **poll()** call, return events will be set to indicate that the event has occurred.

Reading Event Data

Event data in AIX Event Infrastructure consists of keyword-value pairs.

Event data may only be read once and no more than one event occurrence worth of data will be returned in a single **read()** call. For example, say that two events have occurred before the consumer reads from the monitor file and each event has 256 bytes worth of data. If the consumer calls **read()** for 4096 bytes, only the 256 bytes of the first event will be returned to the user. A second **read()** call will need to be performed to obtain the data from the second event. Any offset given will be ignored and data will be read starting from the last unread byte.

Event data will be at the most 4096 bytes, although most events will be much smaller (< 512 bytes). It is recommended that when reading events, a large enough buffer should be used so as to avoid only reading part of an event.

Event data in AHAFS consists of **keyword = value** pairs, with the exception of **BUF_WRAP**, **EVENT_OVERFLOW**, **BEGIN_EVENT_INFO**, **END_EVENT_INFO**, **BEGIN_EVPROD_INFO**, **END_EVPROD_INFO** and **STACK_TRACE** which are special keywords without any values. Here are the keywords you may see in event data:

Table 67. Keywords

Key	Value	Details
BUF_WRAP	None	The consumer buffer is handled like a circular buffer. If any unread data is overwritten by the latest event data, this keyword will be the next string returned by the **read()** call even if the consumer has partially read the previous entry. The subsequent call to **read()** will return the next whole event.
EVENT_OVERFLOW	None	If the event data is too large to fit inside the consumer's event data buffer, this keyword will be returned from the first **read()**. A subsequent **read()** will return what event data was able to fit inside the buffer. **Note:** If **EVENT_OVERFLOW** is encountered, the end string **END_EVENT_INFO** will not be present.
BEGIN_EVENT_INFO	None	This keyword signifies the beginning of the data for an event occurrence.

Table 67. Keywords (continued)

Key	Value	Details
END_EVENT_INFO	None	This keyword signifies the end of the data for this specific event occurrence.
TIME_tvsec TIME_tvnsec	Integer	These two fields record the timestamp of the event occurrence as seconds and nano-seconds since the Epoch.
SEQUENCE_NUM	Integer	This field records the number of times the event has occurred since the first successful **select()** or blocking **read()**. This number is reset to 0 if the **select()** or blocking **read()** call fails, or if the consumer ceases to monitor the event (through overwriting of the event monitoring specifications, or through reaching an event occurrence count equal to the **NOTIFY_CNT** specified).
PID	Integer	Process ID of the process which triggered the event occurrence. Only available with an event producer who has specified the **AHAFS_STKTRACE_AVAILABLE** capability, but not the **AHAFS_CALLBACK_INTRCNTX** capability.
UID	Integer	Effective user ID of the user who triggered the event occurrence. Only available with an event producer who has specified the **AHAFS_STKTRACE_AVAILABLE** capability, but not the **AHAFS_CALLBACK_INTRCNTX** capability.
UID_LOGIN	Integer	Login user ID of the user who triggered the event occurrence. Only available with an event producer who has specified the **AHAFS_STKTRACE_AVAILABLE** capability, but not the **AHAFS_CALLBACK_INTRCNTX** capability.

Table 67. Keywords (continued)		
Key	**Value**	**Details**
GID	Integer	Effective group ID of the user who triggered the event occurrence. Only available with an event producer who has specified the **AHAFS_STKTRACE_AVAILABLE** capability, but not the **AHAFS_CALLBACK_INTRCNTX** capability.
PROG_NAME	String	Name of the process which triggered the event occurrence. Only available with an event producer who has specified the **AHAFS_STKTRACE_AVAILABLE** capability, but not the **AHAFS_CALLBACK_INTRCNTX** capability.
CURRENT_VALUE	64 bit unsigned integer, in decimal	This key is only for **AHAFS_THRESHOLD_VALUE** event producers and will return the current value of the event at the time the event occurrence was detected. Note that due to delay between the time a process is notified and the time they read the event data, the actual current value of the event may have changed.
RC_FROM_EVPROD	32 bit integer, in decimal	This return code comes from the event producer either as the result of an error when trying to set up the event, or as the result of an event occurrence. Generally, return codes less than 256 indicate an error when attempting to register the event with the event producer. Some event producers will return codes greater than 256 to provide more information about the event occurrence. These return codes are documented in **sys/ahafs_evProds.h**

Table 67. Keywords (continued)

Key	Value	Details
BEGIN_EVPROD_INFO **END_EVPROD_INFO**	String*	These two keywords mark the beginning and end of the string returned by the event producer. There will always be a newline after **BEGIN_EVPROD_INFO** and before **END_EVPROD_INFO**. For consumers who have specified **CLUSTER=YES**, this is where node information will be given.
STACK_TRACE	String*	For consumers who have specified **INFO_LVL=3** with an event producer who has specified the **AHAFS_STKTRACE_AVAILABLE** capability, but not the **AHAFS_CALLBACK_INTRCNTX** capability, the stack trace of the event occurrence will be provided. The keyword **STACK_TRACE** indicates that the remaining event data, until the string **END_EVENT_INFO** is the stack of the event occurrence.
NUM_EVDROPS_INTRCNTX	Integer	This keyword represents the number of interrupt-context event occurrences that are dropped since the time that is specified by the TIME0_tvsec and TIME0_tvnsec keywords in the report. The event occurrences are dropped only when the frequency of the event occurrence is high.
TIME0_tvsec TIME0_tvnsec	Integer	These keywords record the time stamp of the first event-occurrence drop in seconds and nano-seconds since the Epoch. These keywords are reported along with the NUM_EVDROPS_INTRCNTX keyword.

Duplicate Event Consolidation

If the same event occurs multiple times before the consumer has read the data, the duplicate entries will be consolidated into one entry. This consolidation is indicated by non sequential sequence numbers without a corresponding **BUF_WRAP** keyword. The timestamp and sequence numbers will reflect the most recent occurrence of the event.

Example Event Data

For an event producer which has specified **AHAFS_THRESHOLD_STATE** and **AHAFS_STKTRACE_AVAILABLE**, and will pass a message to event consumers, the three levels of output look like this:

INFO_LVL=1	INFO_LVL=2	INFO_LVL=3
BEGIN_EVENT_INFO TIME_tvsec=1269863383 TIME_tvnsec=455993143 SEQUENCE_NUM=0 PID=6947038 UID=0 UID_LOGIN=0 GID=0 PROG_NAME=cat RC_FROM_EVPROD=1000 END_EVENT_INFO	BEGIN_EVENT_INFO TIME_tvsec=1269863383 TIME_tvnsec=455993143 SEQUENCE_NUM=0 PID=6947038 UID=0 UID_LOGIN=0 GID=0 PROG_NAME=cat RC_FROM_EVPROD=1000 BEGIN_EVPROD_INFO event producer message here END_EVPROD_INFO END_EVENT_INFO	BEGIN_EVENT_INFO TIME_tvsec=1269863383 TIME_tvnsec=455993143 SEQUENCE_NUM=0 PID=6947038 UID=0 UID_LOGIN=0 GID=0 PROG_NAME=cat RC_FROM_EVPROD=1000 BEGIN_EVPROD_INFO event producer message here END_EVPROD_INFO STACK_TRACE ahafs_prod_callback+3C4 ahafs_cbfn_wrapper+30 ahafs_vn_write+204 vnop_rdwr+7E4 vno_rw+B4 rwuio+12C rdwr+184 kewrite+16C .svc_instr write+1A4 _xwrite+6C _xflsbuf+B0 __flsbuf+9C copyopt_ascii+2C0 scat+388 main+11C __start+68 END_EVENT_INFO

For an event producer which has specified **AHAFS_THRESHOLD_VALUE_HI** and has not specified **AHAFS_STKTRACE_AVAILABLE**, and will pass a message to event consumers, the three levels of output look like this:

INFO_LVL=1	INFO_LVL=2	INFO_LVL=3
BEGIN_EVENT_INFO TIME_tvsec=1269866715 TIME_tvnsec=16678418 SEQUENCE_NUM=0 CURRENT_VALUE=3 RC_FROM_EVPROD=1000 END_EVENT_INFO	BEGIN_EVENT_INFO TIME_tvsec=1269866715 TIME_tvnsec=16678418 SEQUENCE_NUM=0 CURRENT_VALUE=3 RC_FROM_EVPROD=1000 BEGIN_EVPROD_INFO event producer message here END_EVPROD_INFO END_EVENT_INFO	BEGIN_EVENT_INFO TIME_tvsec=1269866715 TIME_tvnsec=16678418 SEQUENCE_NUM=0 CURRENT_VALUE=3 RC_FROM_EVPROD=1000 BEGIN_EVPROD_INFO event producer message here END_EVPROD_INFO END_EVENT_INFO

Error format

If there is an error from the event producer, all event producers will have the following format for all **INFO_LVL**:

```
BEGIN_EVENT_INFO
TIME_tvsec=1269868036
TIME_tvnsec=966708948
SEQUENCE_NUM=0
RC_FROM_EVPROD=20
END_EVENT_INFO
```

If a consumer is monitoring a **AHAFS_THRESHOLD_VALUE** event and the current value already exceeds the requested threshold, the error format will also be used to record this **EALREADY** event:

```
BEGIN_EVENT_INFO
TIME_tvsec=1269868036
TIME_tvnsec=966708948
SEQUENCE_NUM=0
CURRENT_VALUE=1
RC_FROM_EVPROD=56
END_EVENT_INFO
```

BUF_WRAP and EVENT_OVERFLOW

If unread data is overwritten by data from a new event occurrence, the keyword **BUF_WRAP** will be the first output from a **read()** on the monitor file. If there is a buffer wrap AND an event overflow, the **BUF_WRAP** will always come first, followed by the **EVENT_OVERFLOW**. Here is an example output from **read()** in the case where we have both a buffer wrap and an event overflow:

First **read()** will return:

```
BUF_WRAP
```

Second **read()** will return:

```
EVENT_OVERFLOW
```

Third **read()** will return the event data that was able to fit inside the buffer:

```
BEGIN_EVENT_INFO
TIME_tvsec=1269863383
TIME_tvnsec=455993143
SEQUENCE_NUM=0
PID=6947038
UID=0
UID_LOGIN=0
GID=0
PROG_NAME=cat
RC_FROM_EVPROD=1000
BEGIN_EVPROD_INFO
event producer message here
END_EVPROD_INFO
STACK_TRACE
ahafs_prod_callback+3C4
ahafs_cbfn_wrapper+30
ahafs_vn_write+204
vnop_rdwr+7E4
vno_rw+B4
rwuio+12C
rdwr+184
kewrite+16C
.svc_instr
write+1A4
_xwri
```

If event information is coming fast enough, it is possible to receive two **BUF_WRAP** entries in a row. If you are seeing **BUF_WRAP**, increase the size of the buffer (using **BUF_SIZE** when writing to the monitor file).

NUM_EVDROPS_INTRCNTX

If any interrupt-context event occurrence is dropped because of a high frequency of event-occurrences, the output from a **`read()`** call on the event file, representing that event, contains the `NUM_EVDROPS_INTRCNTX` keyword just after the line that contains the `BEGIN_EVENT_INFO` keyword.

The following example represents an output from a **`read()`** call:

```
BEGIN_EVENT_INFO
BEGIN_EVENT_INFO
NUM_EVDROPS_INTRCNTX=5508
TIME0_tvsec=1353437661
TIME0_tvnsec=75494625
TIME_tvsec=1353437661
TIME_tvnsec=741365037
```

```
SEQUENCE_NUM=6663
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
...msg from event-producer...
END_EVPROD_INFO
END_EVENT_INFO
```

This example output contains the following sets of information:

- The NUM_EVDROPS_INTRCNTX=5508 value is the number of dropped interrupt-context event-occurrences since the time that is specified by the TIME0_tvsec and TIME0_tvnsec fields.
- The remaining information (that is, SEQUENCE_NUM=6663, RC_FROM_EVPROD=0, ...msg from event-producer..., and so on) is about the event that occurred at the time that is specified by the TIME_tvsec and TIME_tvnsec fields.

Pre-defined event producers

modFile

The **modFile** event producer monitors for modifications to the contents of a file.

Overview

The **modFile** event producer resides under the **fs** directory and monitors for modifications to a file. The following **vnode** operations are monitored: **vnop_rdwr()**, **vnop_map_lloff()**, **vnop_remove()**, **vnop_ftrunc()**, **vnop_fclear()** and **vnop_rename()**. Modifications which do not go through the LFS layer cannot be monitored (that is writes to mapped files).

Files may not be monitored if:

- They are in a remote file system.
- They are in file system of type **ahafs**, **procfs** or **namefs**.
- They are a symbolic link.
- They reside under a directory which ends with an AIX Event Infrastructure extension (**.mon**, **.list**, **.monFactory**).
- Monitor files with a full path name larger than **MAXPATHLEN** in the AIX Event Infrastructure pseudo file system cannot be monitored.

Capabilities

```
AHAFS_THRESHOLD_STATE
AHAFS_STKTRACE_AVAILABLE
AHAFS_REMOTE_EVENT_ENABLED
```

Return codes

The **modFile** event producer uses return codes which are defined in **<sys/ahafs_evProds.h>**.

These return codes are used to indicate how the contents of the monitored directory were modified:

AHAFS_MODFILE_WRITE
The monitored file was written to.

AHAFS_MODFILE_UNMOUNT
The file system containing the monitored file was unmounted. This is an unavailable event.

AHAFS_MODFILE_MAP
A process has mapped a portion of the monitored file for writing.

AHAFS_MODFILE_REMOVE
The monitored file has been removed. This is an unavailable event.

AHAFS_MODFILE_RENAME
The monitored file has been renamed. This is an unavailable event.

AHAFS_MODFILE_FCLEAR
A process has issued an **fclear** for the monitored file.

AHAFS_MODFILE_FTRUNC
A process has issued an **ftrunc** for the monitored file.

AHAFS_MODFILE_OVERMOUNT
The monitored file has been over mounted.

Event producer message

This event producer does not pass any messages as part of its event data.

Acceptable monitor files

To monitor for file modifications, a monitor file with the same path as the file you wish to monitor should be created under the **modFile.monFactory** directory. For example, to monitor **/etc/passwd**, the monitor file **/aha/fs/modFile.monFactory/etc/passwd.mon** would be used.

Example event data

The following event data was generated from a process writing to a monitored file. This is the output seen with an **INFO_LVL** of 3:

```
BEGIN_EVENT_INFO
TIME_tvsec=1271703118
TIME_tvnsec=409201093
SEQUENCE_NUM=0
PID=5701678
UID=0
UID_LOGIN=0
GID=0
PROG_NAME=cat
RC_FROM_EVPROD=1000
STACK_TRACE
aha_cbfn_wrapper+30
ahafs_evprods+510
aha_vn_write+154
vnop_rdwr+7E8
vno_rw+B4
rwuio+100
rdwr+188
kewrite+104
.svc_instr
write+1A4
_xwrite+6C
_xflsbuf+A8
__flsbuf+C0
copyopt+2E8
scat+22C
main+11C
__start+68
END_EVENT_INFO
```

modFileAttr

The **modFileAttr** event producer monitors for modifications to the attributes of a file.

Overview

The **modFileAttr** event producer resides under the **fs** directory and monitors for modifications to the attributes of a file or directory (mode, ownership and ACLs). The following vnode operations are monitored: **vnop_setattr()** (only for **V_OWN** and **V_MODE** operations), **vnop_setacl()**, **vnop_setxacl()**, **vnop_remove()**, **vnop_rename()** and **vnop_rmdir()**.

Files or directories may not be monitored if:

- They are in a remote file system
- They are in file system of type **ahafs**, **procfs** or **namefs**
- They reside under a directory which ends with an AIX Event Infrastructure extension (**.mon, .list, .monFactory**)

- Monitor files with a full path name larger than **MAXPATHLEN** in the AIX Event Infrastructure pseudo file system cannot be monitored.

Capabilities

```
AHAFS_THRESHOLD_STATE

AHAFS_STKTRACE_AVAILABLE

AHAFS_REMOTE_EVENT_ENABLED
```

Return codes

The **modFileAttr** event producer uses return codes which are defined in **<sys/ahafs_evProds.h>**.

These return codes are used to indicate how the contents of the monitored directory were modified:

AHAFS_MODFILEATTR_UNMOUNT
The filesystem containing the monitored file or directory was unmounted. This is an unavailable event.

AHAFS_MODFILEATTR_REMOVE
The monitored file or directory has been removed. This is an unavailable event.

AHAFS_MODFILEATTR_RENAME
The monitored file or directory has been renamed. This is an unavailable event.

AHAFS_MODFILEATTR_OVERMOUNT
The monitored file or directory has been over mounted. This is an unavailable event.

AHAFS_MODFILEATTR_SETACL
The ACLs of the monitored file or directory were modified.

AHAFS_MODFILEATTR_SETOWN
The ownership of the monitored file or directory was modified.

AHAFS_MODFILEATTR_SETMODE
The mode of the monitored file or directory was modified.

Event producer message

This event producer does not pass any messages as part of its event data.

Acceptable monitor files

To monitor for file modifications, a monitor file with the same path as the file you wish to monitor should be created under the **modFileAttr.monFactory** directory. For example, to monitor **/etc/passwd**, the monitor file **/aha/fs/modFileAttr.monFactory/etc/passwd.mon** would be used.

Example event data

The following event data was generated from a process changing the mode of a monitored file. This is the output seen with an **INFO_LVL** of 3:

```
BEGIN_EVENT_INFO
TIME_tvsec=1291994430
TIME_tvnsec=760097298
SEQUENCE_NUM=0
PID=5767216
UID=0
UID_LOGIN=0
GID=0
PROG_NAME=chmod
RC_FROM_EVPROD=1010
STACK_TRACE
ahafs_evprods+70C
aha_process_attr+120
vnop_setattr+21C
vsetattr@AF13_1+20
setnameattr+B4
chmod+110
.svc_instr
change+3C8
main+190
```

```
__start+68
END_EVENT_INFO
```

modDir

The **modDir** event producer monitors for modifications to the contents of a directory.

Overview

The **modDir** event producer resides under the **fs** directory and monitors for modifications to the contents of a directory. The following **vnode** operations are monitored: **vnop_create()**, **vnop_link()**, **vnop_symlink()**, **vnop_remove()**, **vnop_rename()**, **vnop_mkdir()**, and **vnop_rmdir()**.

Directories may not be monitored if:

- They are in a remote file system
- They are in file system of type **ahafs**, **procfs** or **namefs**
- They are a symbolic link
- They reside under a directory which ends with an AIX Event Infrastructure extension (**.mon**, **.list**, **.monFactory**)
- Monitor files with a full path name larger than **MAXPATHLEN** in the AIX Event Infrastructure pseudo file system cannot be monitored.

The **modDir** event producer does not recursively monitor for directory modifications. Only modifications to the specified directory are monitored.

Capabilities

```
AHAFS_THRESHOLD_STATE
AHAFS_STKTRACE_AVAILABLE
AHAFS_REMOTE_EVENT_ENABLED
```

Return codes

The **modDir** event producer uses return codes which are defined in **<sys/ahafs_evProds.h>**.

These return codes are used to indicate how the contents of the monitored directory were modified:

AHAFS_MODDIR_CREATE
A new file system object has been created under the monitored directory.

AHAFS_MODDIR_UNMOUNT
The file system containing the monitored directory has been unmounted. This is an unavailable event.

AHAFS_MODDIR_REMOVE
A file system object within the monitored directory has been removed.

AHAFS_MODDIR_REMOVE_SELF
The monitored directory itself has been removed or renamed. This is an unavailable event.

Event producer message

The name of the file system object which triggered the event is included in the event data.

Acceptable monitor files

To monitor for modifications to the contents of a directory, a monitor file with the same path as the directory you wish to monitor should be created under the **modDir.monFactory** directory. For example, to monitor the directory **/home/cheryl** for modifications, the monitor file **/aha/fs/modDir.monFactory/home/cheryl.mon** would be used.

Example event data

The following event data was generated from a new file named **file1** being created in a monitored directory. This is the output seen with an **INFO_LVL** of 3:

```
BEGIN_EVENT_INFO
TIME_tvsec=1271780397
TIME_tvnsec=24369022
SEQUENCE_NUM=0
PID=6095102
UID=0
UID_LOGIN=0
GID=0
PROG_NAME=touch
RC_FROM_EVPROD=1000
BEGIN_EVPROD_INFO
file1
END_EVPROD_INFO
STACK_TRACE
aha_cbfn_wrapper+30
ahafs_evprods+510
aha_process_vnop+138
vnop_create_attr+4AC
openpnp+418
openpath+100
copen+294
kopen+1C
.svc_instr
open+F8
creat64+1C
main+1EC
__start+68
END_EVENT_INFO
```

utilFs

The **utilFs** event producer monitors the utilization of a file system.

Overview

The **utilFs** event producer monitors the utilization of a file system as a percentage. It resides under the **fs** directory. Currently, only JFS2 file systems support **utilFs** monitoring. Upon every file write, file creation and file deletion, the utilization of the file system is checked to see if it meets or exceeds the given threshold. There may be some file system specific operations which can affect the utilization of the file system, but **utilFs** may not be able to detect them until the next file write, creation or deletion. Thresholds which are exceeded due to the result of a file object deletion will not be notified until the next file write, create or deletion.

File systems with a monitor file path name larger than **MAXPATHLEN** in AHAFS cannot be monitored.

To avoid a flood of event notifications and potential performance impacts, it is highly recommended that **utilFs** events are monitored with a **NOTIFY_CNT** of 1.

Capabilities

```
AHAFS_THRESHOLD_VALUE_HIGH
AHAFS_THRESHOLD_VALUE_LOW
AHAFS_REMOTE_EVENT_ENABLED
```

Thresholds specified must be between 1 and 100, inclusive.

Return codes

The **utilFs** event producer uses return codes which are defined in **<sys/ahafs_evProds.h>**.

These return codes are used to indicate how the contents of the monitored directory were modified:

AHAFS_UTILFS_THRESH_HIT
The file system being monitored has reached the threshold specified.

AHAFS_UTILFS_UNMOUNT
The file system being monitored has been unmounted. This is an unavailable event.

Event producer message

This event producer does not pass any messages as part of its event data.

Acceptable monitor files

To monitor for file system utilization, a monitor file with the same path as the mount point of the file system to be monitored should be created under the **utilFs.monFactory** directory. For example, to monitor the file system **/data/fs1**, the monitor file **/aha/fs/utilFs.monFactory/data/fs1.mon** would be used.

Example event data

The following is event data from an **AHAFS_UTILFS_THRESH_HIT** event for an **INFO_LVL** of 3:

```
BEGIN_EVENT_INFO
TIME_tvsec=1271705858
TIME_tvnsec=704241888
SEQUENCE_NUM=0
CURRENT_VALUE=10
RC_FROM_EVPROD=1000
END_EVENT_INFO
```

waitTmCPU

The **waitTmCPU** event producer monitors the average wait time of runnable threads.

Overview

The **waitTmCPU** event producer monitors the average wait time of runnable threads waiting to get CPU time in one second intervals, measured in milliseconds. The **waitTmCPU** resides under the **cpu** directory.

Capabilities

```
AHAFS_THRESHOLD_VALUE_HIGH
AHAFS_CALLBACK_INTRCNTX
AHAFS_REMOTE_EVENT_ENABLED
```

Thresholds specified must be greater than 0.

Return codes

This event producer always returns 0 when the event occurs.

Event producer message

This event producer does not pass any messages as part of its event data.

Acceptable monitor files

To monitor this event, the following monitor file should be used:

```
/aha/cpu/waitTmCPU.monFactory/waitTmCPU.mon
```

No other monitor files may be created in this directory.

Example event data

The following is event data from a **waitTmCPU** event with an **INFO_LVL** of 3:

```
BEGIN_EVENT_INFO
TIME_tvsec=1271779504
TIME_tvnsec=18056777
SEQUENCE_NUM=0
CURRENT_VALUE=4
RC_FROM_EVPROD=0
END_EVENT_INFO
```

waitersFreePg

The **waitersFreePg** event producer monitors the number of threads waiting for a free frame.

Overview

The **waitersFreePg** event producer monitors the number of threads waiting for a free frame over one second intervals. The **waitersFreePg** resides under the **mem** subdirectory.

Capabilities

```
AHAFS_THRESHOLD_VALUE_HIGH
AHAFS_CALLBACK_INTRCNTX
AHAFS_REMOTE_EVENT_ENABLED
```

Thresholds specified must be greater than 0.

Return codes

This event producer always returns 0 when the event occurs.

Event producer message

This event producer does not pass any messages as part of its event data.

Acceptable monitor files

To monitor this event, the following monitor file should be used:

```
/aha/mem/waitersFreePg.monFactory/waitersFreePg.mon
```

No other monitor files may be created in this directory.

Example event data

The following is event data from a **waitersFreePg** event with an **INFO_LVL** of 3:

```
BEGIN_EVENT_INFO
TIME_tvsec=1271779680
TIME_tvnsec=347233732
SEQUENCE_NUM=0
CURRENT_VALUE=19843
RC_FROM_EVPROD=0
END_EVENT_INFO
```

waitTmPgInOut

The **waitTmPgInOut** event producer monitors for the average wait time in milliseconds for threads waiting for a page in or page out operations.

Overview

The **waitTmPgInOut** event producer monitors for the average wait time in milliseconds for threads waiting for page in or page out operations to complete over a one second period. The **waitTmPgInOut** event producer resides under the **mem** directory.

Capabilities

```
AHAFS_THRESHOLD_VALUE_HIGH
AHAFS_CALLBACK_INTRCNTX
AHAFS_REMOTE_EVENT_ENABLED
```

Thresholds specified must be greater than 0.

Return codes

This event producer always returns 0 when the event occurs.

Event producer message

This event producer does not pass any messages as part of its event data.

Acceptable monitor files

To monitor this event, the following monitor file should be used:

```
/aha/mem/waitTmPgInOut.monFactory/waitTmPgInOut.mon
```

No other monitor files may be created in this directory.

Example event data

The following is event data from a **waitTmPgInOut** event with an **INFO_LVL** of 3:

```
BEGIN_EVENT_INFO
TIME_tvsec=1271779359
TIME_tvnsec=941699413
SEQUENCE_NUM=0
CURRENT_VALUE=12
RC_FROM_EVPROD=0
END_EVENT_INFO
```

vmo

The **vmo** event producer monitors for changes to the **vmo** tunable parameters.

Overview

The **vmo** event producer resides under the **mem** directory and monitors for changes to the following **vmo** tunables.

Note: The **vmo** command is a self documenting command. Some of the tunable parameters listed in the following list might not be supported.

- **npskill**
- **npswarn**
- **force_relalias_lite**
- **low_ps_handling**
- **maxpin%** (should be monitored as **maxpin_pct.mon** file)
- **nokilluid**
- **relalias_percentage**
- **vmm_default_pspa**
- **npsrpgmin**
- **npsrpgmax**
- **npsscrubmin**
- **npsscrubmax**
- **scrubclean**
- **rpgcontrol**
- **rpgclean**
- **vm_modlist_threshold**
- **vmm_fork_policy**
- **lru_poll_interval**

Capabilities

```
AHAFS_THRESHOLD_STATE
AHAFS_STKTRACE_AVAILABLE
AHAFS_REMOTE_EVENT_ENABLED
```

Return codes

This event producer always returns 0 when the event occurs.

Event producer message

This event producer does not pass any messages as part of its event data.

Acceptable monitor files

To monitor any of the above tunables, monitor files of the following format should be used:

```
/aha/mem/vmo.monFactory/<tunable>.mon
```

Files which do not correspond to the above events cannot be created under this directory.

Example event data

The following is event data from the modification of a monitored tunable, with an **INFO_LVL** of 3.

```
BEGIN_EVENT_INFO
TIME_tvsec=1271770698
TIME_tvnsec=787565808
SEQUENCE_NUM=0
PID=5701808
UID=0
UID_LOGIN=0
GID=0
PROG_NAME=vmo
RC_FROM_EVPROD=0
STACK_TRACE
aha_cbfn_wrapper+30
ahafs_evprods+510
vm_mon_tunable+B0
vm_chk_mod_tun+5CC
_vmgetinfo+53C
vmgetinfo+48
.svc_instr
vmo_write_vmsetkervars+134
vmo_write_dynamic_values+404
main+BC
__start+70
END_EVENT_INFO
```

schedo

This event producer monitors for changes to **schedo** tunables.

Overview

Currently, only the **vpm_xvcpus** tunable may be monitored. This event producer will return a stack trace and user information when the event occurs. This event producer resides under the **cpu** directory.

Capabilities

```
AHAFS_THRESHOLD_STATE
AHAFS_STKTRACE_AVAILABLE
AHAFS_REMOTE_EVENT_ENABLED
```

Return codes

This event producer always returns 0 when the event occurs.

Event producer message

This event producer does not pass any messages as part of its event data.

Acceptable monitor files

The monitor file used to monitor this tunable is:

```
/aha/cpu/schedo.monFactory/vpm_xvcpus.mon
```

No other monitor files may be created in this directory.

Example event data

The following is event data from the modification of the **vpm_xvcpus** tunable with an **INFO_LVL** of 3:

```
BEGIN_EVENT_INFO
TIME_tvsec=1271771009
TIME_tvnsec=251723285
SEQUENCE_NUM=0
PID=7143474
UID=0
UID_LOGIN=0
GID=0
PROG_NAME=schedo
RC_FROM_EVPROD=0
STACK_TRACE
aha_cbfn_wrapper+30
ahafs_evprods+510
schedtune+394
.svc_instr
schedo_write_schedparams+94
schedo_write_dynamic_values+6F0
main+1B0
__start+68
END_EVENT_INFO
```

pidProcessMon

The **pidProcessMon** event producer monitors for process death, based on PID.

Overview

The **pidProcessMon** event producer resides under the **cpu** directory and monitors for process death, based on PID.

Capabilities

```
AHAFS_THRESHOLD_STATE
AHAFS_CALLBACK_INTRCNTX
```

Return codes

The **pidProcessMon** event producer returns only a single return code 0.

Event producer message

This event producer passes **PROCESS_DOWN** message as part of its event data.

Acceptable monitor files

To monitor for process deaths, a monitor file should be created under the **pidProcessMon.monFactory** directory. A monitor file name with the format

```
/aha/cpu/pidProcessMon.monFactory/<process_PID>.mon
```

has to be used.

Example event data

The following event data was generated from the death of a monitored process. This is the output seen with the default **INFO_LVL**.

```
BEGIN_EVENT_INFO
TIME_tvsec=1272348759
TIME_tvnsec=379259175
SEQUENCE_NUM=0
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=PROCESS_DOWN
END_EVPROD_INFO
END_EVENT_INFO
```

processMon

The **processMon** event producer monitors for process death.

Overview

The **processMon** event producer resides under the cpu directory and monitors for process death, based on process name. Only the parent process for a given process with same name is monitored. That means if we have a process tree **abc (pid 123)->xyz (pid 345)->xyz (pid 567)** and some one requests to monitor the **xyz** process then (**pid = 345**) actually gets monitored.

Capabilities

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

Return codes

The **processMon** event producer returns only a single return code 0.

Event producer message

This event producer passes **PROCESS_DOWN** message as part of its event data.

Acceptable monitor files

To monitor for process deaths, a monitor file with the same path as the one used to start the process, should be created under the **processMon.monFactory** directory. For example, to monitor a process named **test** which is placed under the directory **/usr/samples/ahafs**, the monitor file **/aha/cpu/processMon.monFactory/usr/samples/ahafs/test.mon** would be used.

Example event data

The following event data was generated from the death of a monitored process. This is the output seen with the default **INFO_LVL**.

```
BEGIN_EVENT_INFO
TIME_tvsec=1272348909
TIME_tvnsec=482502597
SEQUENCE_NUM=0
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=PROCESS_DOWN
END_EVPROD_INFO
END_EVENT_INFO
```

inetsock

The `inetsock` event producer monitors Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) socket operations.

Overview

The `inetsock` event producer is placed under the net directory and monitors socket operations.

The following socket operations are monitored for TCP:

- Creating a socket
- Binding a port or address to the socket
- Listening on the socket
- Accepting and establishing a connection on the socket
- Connecting to a socket
- Disconnecting from a socket
- Closing the socket

The following socket operations are monitored for UDP:

- Creating a socket
- Binding a port or address to the socket

- Closing the socket

Capabilities

```
AHAFS_THRESHOLD_STATE
AHAFS_CALLBACK_INTRCNTX
AHAFS_REMOTE_EVENT_ENABLED
```

Event producer message

This event producer passes information that is available in the protocol control block and socket as a part of its event data.

The following data is passed for the TCP socket operations:

Socket operation	Data
PRU_ATTACH	Common information: • PROG_NAME • SO_FAMILY • SO_PID • SO_PROTO • SO_TYPE • SO_UID
PRU_BIND	Common information plus the following items: • SO_LADDR • SO_LPORT
PRU_LISTEN	Common information plus the following items: • SO_LADDR • SO_LPORT
PRU_ACCEPT	Common information plus the following items: • SO_FADDR • SO_FPORT • SO_LADDR • SO_LPORT
PRU_CONNECT	Common information plus the following items: • SO_FADDR • SO_FPORT • SO_LADDR • SO_LPORT
PRU_DISCONNECT	Common information plus the following items: • SO_FADDR • SO_FPORT • SO_LADDR • SO_LPORT

Socket operation	Data
PRU_DETACH, PRU_ABORT	Common information plus the following items: • SO_LADDR • SO_LPORT • SO_FADDR • SO_FPORT

The following data is passed for the UDP socket operations:

Socket operation	Data
PRU_ATTACH	Common information: • PROG_NAME • SO_FAMILY • SO_PID • SO_PROTO • SO_TYPE • SO_UID
PRU_BIND, PRU_DYNBIND	Common information plus the following items: • SO_LADDR • SO_LPORT
PRU_DETACH, PRU_ABORT	Common information plus the following items: • SO_LADDR • SO_LPORT

Acceptable monitor files

To monitor the socket operations, a monitor file that has the name of the socket operation that you want to monitor must be created in the `inetsock.monFactory` directory. For example, to monitor the TCP socket creation, the `/aha/net/inetsock.monFactory/streamCreate.mon` monitor file is used. Similarly, to monitor UDP socket creation, the `/aha/net/inetsock.monFactory/dgramCreate.mon` monitor file is used.

The following files are used for all the Autonomic Health Advisor File System (AHAFS)-monitorable TCP socket operations:

- `/aha/net/inetsock.monFactory/streamCreate.mon`
- `/aha/net/inetsock.monFactory/streamBind.mon`
- `/aha/net/inetsock.monFactory/streamListen.mon`
- `/aha/net/inetsock.monFactory/streamAccept.mon`
- `/aha/net/inetsock.monFactory/streamConnect.mon`
- `/aha/net/inetsock.monFactory/streamDisconnect.mon`
- `/aha/net/inetsock.monFactory/streamClose.mon`

The following files are used for all the AHAFS-monitorable UDP socket operations:

- `/aha/net/inetsock.monFactory/dgramCreate.mon`
- `/aha/net/inetsock.monFactory/dgramBind.mon`
- `/aha/net/inetsock.monFactory/dgramClose.mon`

Example event data

The following event data was generated from a process when a socket is created. The following example is the output that is displayed with an output level of 2 (INFO_LVL=2):

```
AHAFS event: /aha/net/inetsock.monFactory/streamCreate.mon
---------------------------------------------------

BEGIN_EVENT_INFO
Time          : Mon Jan 23 23:04:06 2012
Sequence Num: 1
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
PROG_NAME=xmtopas
SO_FAMILY=2
SO_TYPE=1
SO_PROTO=6
SO_UID=0
SO_PID=5243048
END_EVPROD_INFO
END_EVENT_INFO
```

Cluster events

When a system is part of a cluster, it can receive notifications for events occurring on other nodes that are part of the same cluster. Event producers which specify the **AHAFS_REMOTE_EVENT_ENABLED** capability support cluster wide monitoring. All the AIX Event Infrastructure event producers except the **pidProcessMon** and **diskState** can provide such remote notifications.

Behaviour of **mkcluster** command with the AIX Event Infrastructure:

If the AIX Event Infrastructure is not loaded on a system and the **mkcluster** command is run then the AIX Event Infrastructure pseudo filesystem will be mounted on the **/aha** directory and the monitor file names will start from the **/aha** directory. If the AIX Event Infrastructure is already loaded on a system and **mkcluster** command is run then the AIX Event Infrastructure pseudo filesystem will not be remounted and the monitor file names will start from the directory over which the AIX Event Infrastructure pseudo filesystem has been mounted. Consumer applications must check where the AIX Event Infrastructure pseudo filesystem has been mounted, to get the monitor file paths.

In order to receive cluster events, consumer processes must specify **CLUSTER=YES** when writing to the monitor file representing the event to monitor in the cluster. In order for the remote events to be detected, a consumer process must be monitoring the event on each node with **CLUSTER=YES** specified.

Events received from a remote node do not include user or process information, or stack trace, even if the event producer supports it.

For events received on a remote node stack trace is not provided, even if the event producer supports it.

The cluster information **NODE_NUMBER**, **NODE_ID** and **CLUSTER_ID** will be available between **BEGIN_EVPROD_INFO** and **END_EVPROD_INFO** delimiters for all cluster events. This helps the monitoring program to identify on which node the event occurred. The information that is returned from the **lscluster -m** command output in the fields: Cluster shorthand id for node, **uuid** for node and cluster **uuids** is returned in the AIX Event Infrastructure event output in the **NODE_NUMBER**, **NODE_ID**, **CLUSTER_ID** fields respectively.

Below is example output from both a local and remote occurrence of an event with an **INFO_LVL** of 2, and an event producer which specifies the **AHAFS_STKTRACE_AVAILABLE** capability.

Local Event Data	Remote Event Data
BEGIN_EVENT_INFO TIME_tvsec=1262670289 TIME_tvnsec=453840229 SEQUENCE_NUM=0 PID=4194474 UID=0 UID_LOGIN=0 GID=0 PROG_NAME=rpc.statd RC_FROM_EVPROD=0 BEGIN_EVPROD_INFO NODE_NUMBER=1 NODE_ID=0xF079E8C801C11DF CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404 EVENT_TYPE=PROCESS_DOWN END_EVPROD_INFO END_EVENT_INFO	BEGIN_EVENT_INFO TIME_tvsec=1262670289 TIME_tvnsec=248144872 SEQUENCE_NUM=0 RC_FROM_EVPROD=0 BEGIN_EVPROD_INFO EVENT_TYPE=PROCESS_DOWN NODE_NUMBER=1 NODE_ID=0xF079E8C801C11DF CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404 END_EVPROD_INFO END_EVENT_INFO

Pre-defined event producers for a Cluster Aware AIX instance

These event producers are only available when the system is part of an active cluster.

nodeList

The **nodeList** event producer monitors changes in cluster membership.

Overview

The **nodeList** event producer resides under the cluster directory and monitors for nodes added or removed from the cluster. This event producer is available only when the system is part of a cluster. This event is generated when a node is added or removed from the cluster (for example, via the **chcluster** command).

Capabilities

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

Return codes

The **nodeList** returns 0 as the return code. Only if the cluster is removed then **AHAFS_CLUSTER_REMOVE (-1)** is returned.

Event producer message

This event producer passes **NODE_ADD** and **NODE_DELETE** messages as part of its event data. Also, as it is a cluster event producer it will additionally pass **NODE_NUMBER**, **NODE_ID** and **CLUSTER_ID** information.

Acceptable monitor files

To monitor for changes in the list of nodes, a monitor file should be created under the **nodeList.monFactory** directory. The monitor file name

```
/aha/cluster/nodeList.monFactory/nodeListEvent.mon
```

has to be used. No other monitor files may be created in this directory.

Example event data

The following is event data from a **nodeList** event with the default **INFO_LVL**.

```
BEGIN_EVENT_INFO
TIME_tvsec=1271922590
TIME_tvnsec=886742634
SEQUENCE_NUM=1
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=NODE_ADD
```

```
NODE_NUMBER=1
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

clDiskList

The **clDiskList** event producer monitors changes in cluster membership.

Overview

The **clDiskList** event producer resides under the disk directory and monitors for disks added or removed from the cluster. This event producer is available only when the system is part of a cluster. This event is generated when a disk is added or removed from the cluster (for example, using the **chcluster** command).

Capabilities

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

Return codes

The **clDiskList** returns 0 as the return code. Only if the cluster is removed **AHAFS_CLUSTER_REMOVE (-1)** is returned.

Event producer message

This event producer passes the **DISK_ADD** and **DISK_DELETE** messages as part of its event data in the **EVENT_TYPE** field. It will pass the **DISK_NAME** and the **DISK_UID** of the concerned disk. Also, as it is part of a cluster event producer it will additionally pass the **NODE_NUMBER**, **NODE_ID** and **CLUSTER_ID** information.

Acceptable monitor files

To monitor for changes in the list of disks, a monitor file should be created under the **clDiskList.monFactory** directory. The monitor file name

```
/aha/disk/clDiskList.monFactory/clDiskListEvent.mon
```

has to be used. No other monitor files may be created in this directory.

Example event data

The following is event data from a **clDiskList** event with the default **INFO_LVL**.

```
BEGIN_EVENT_INFO
TIME_tvsec=1271927983
TIME_tvnsec=696543410
SEQUENCE_NUM=0
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=DISK_ADD
DISK_NAME=cldisk1
DISK_UID=3E213600A0B800016726C000000FF4B8677C80F1724-100 FAStT03IBMfcp
NODE_NUMBER=2
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

linkedCl

The **linkedCl** event producer is generated when a cluster is linked or unlinked with another cluster.

Overview

The **linkedCl** event producer resides under the cluster directory and monitors for link status changes. This event producer is available only when the system is part of a cluster. This event is generated when a cluster is linked or unlinked with another cluster.

Capabilities

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

Return codes

The **linkedCl** returns 0 as the return code. Only if the cluster is removed **AHAFS_CLUSTER_REMOVE (-1)** is returned.

Event producer message

This event producer passes **LINK_UP** or **LINK_DOWN** messages as part of its event data. It will pass the **LINK_ID** information. Also, as it is a cluster event producer it will additionally pass **NODE_NUMBER**, **NODE_ID** and **CLUSTER_ID** information.

Acceptable monitor files

To monitor for changes in the list of nodes, a monitor file should be created under the **linkedCl.monFactory** directory. The monitor file name

```
/aha/cluster/linkedCl.monFactory/linkedClEvent.mon
```

has to be used. No other monitor files may be created in this directory.

Example event data

The following is event data from a **linkedCl** event with the default **INFO_LVL**.

```
BEGIN_EVENT_INFO
TIME_tvsec=1271224025
TIME_tvnsec=795042625
SEQUENCE_NUM=0
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=LINK_DOWN
LINK_ID=0x7BE9C1BD
NODE_NUMBER=1
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

nodeContact

The **nodeContact** event producer monitors the last contact status of the node in a cluster.

Overview

The **nodeContact** event producer resides under the cluster directory and monitors the last contact status of the node in the cluster. This event producer is available only when the system is part of a cluster.

Capabilities

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

Return codes

The **nodeContact** returns 0 as the return code. Only if the cluster is removed **AHAFS_CLUSTER_REMOVE (-1)** is returned.

Event producer message

This event producer passes the **CONNECT_UP** and **CONNECT_DOWN** messages as part of its event data. It will pass the concerned **INTERFACE_NAME**. Also, as it is a cluster event producer it will additionally pass the **NODE_NUMBER**, **NODE_ID** and **CLUSTER_ID** information.

Acceptable monitor files

To monitor for changes in the list of nodes, a monitor file should be created under the **nodeContact.monFactory** directory. The monitor file name

```
/aha/cluster/nodeContact.monFactory/nodeContactEvent.mon
```

has to be used. No other monitor files may be created in this directory.

Example event data

The following is event data from a **nodeContact** event with the default **INFO_LVL**.

```
BEGIN_EVENT_INFO
TIME_tvsec=1271921874
TIME_tvnsec=666770128
SEQUENCE_NUM=0
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=CONNECT_DOWN
INTERFACE_NAME=en0
NODE_NUMBER=2
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

nodeState

The **nodeState** event producer monitors for the state of a node in the cluster.

Overview

The **nodeState** event producer resides under the cluster directory and monitors for the state of a node in the cluster. This event producer is available only when the system is part of a cluster. This event is generated, for example, when a node crashes or is shutdown.

Capabilities

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

Return codes

The **nodeState** returns 0 as the return code. Only if the cluster is removed **AHAFS_CLUSTER_REMOVE (-1)** is returned.

Event producer message

This event producer passes **NODE_UP** and **NODE_DOWN** messages as part of its event data. Also, as it is a cluster event producer and it will additionally pass the **NODE_NUMBER**, **NODE_ID** and **CLUSTER_ID** information.

Acceptable monitor files

To monitor for changes in the status of nodes, a monitor file should be created under the **nodeState.monFactory** directory. The monitor file name

```
/aha/cluster/nodeState.monFactory/nodeStateEvent.mon
```

has to be used. No other monitor files may be created in this directory.

Example event data

The following is event data from a **nodeState** event with the default **INFO_LVL**.

```
BEGIN_EVENT_INFO
TIME_tvsec=1271921536
TIME_tvnsec=68254861
SEQUENCE_NUM=1
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=NODE_UP
NODE_NUMBER=2
NODE_ID=0xF079E8C801C11DF
```

```
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

nodeAddress

The **nodeAddress** event producer monitors the network address of the node.

Overview

The **nodeAddress** event producer resides under the cluster directory and monitors the network address of the node. This event producer is available only when the system is part of a cluster. This event is generated for example, when an alias is added or removed from a network interface.

Capabilities

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

Return codes

The **nodeAddress** returns 0 as the return code. Only if the cluster is removed **AHAFS_CLUSTER_REMOVE (-1)** is returned.

Event producer message

This event producer passes **ADDRESS_ADD** and **ADDRESS_DELETE** messages as part of its event data. It will pass the **INTERFACE_NAME**, of the concerned interface and the **FAMILY**, **ADDRESS** and **NETMASK** of the IP address. Also, as it is a cluster event producer it will additionally pass the **NODE_NUMBER**, **NODE_ID** and **CLUSTER_ID** information.

Acceptable monitor files

To monitor for changes in the list of nodes, a monitor file should be created under the **nodeAddress.monFactory** directory. The monitor file name

```
/aha/cluster/nodeAddress.monFactory/nodeAddressEvent.mon
```

has to be used. No other monitor files may be created in this directory.

Example event data

The following is event data from a **nodeAddress** event with the default **INFO_LVL**.

```
BEGIN_EVENT_INFO
TIME_tvsec=1271922254
TIME_tvnsec=9053410
SEQUENCE_NUM=0
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=ADDRESS_ADD
INTERFACE_NAME=et0
FAMILY=2
ADDRESS=0x0A0A0A0A
NETMASK=0xFF000000
NODE_NUMBER=2
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

networkAdapterState

The **networkAdapterState** event producer monitors the network interface of a node in the cluster.

Overview

The **networkAdapterState** event producer resides under the cluster directory and monitors the network interface of a node in the cluster. This event producer is available only when the system is part of a cluster. This event is generated when a network interface goes down or comes up.

Capabilities

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

Return codes

The **networkAdapterState** returns 0 as the return code. Only if the cluster is removed **AHAFS_CLUSTER_REMOVE (-1)** is returned.

Event producer message

This event producer passes the **ADAPTER_UP**, **ADAPTER_DOWN**, **ADAPTER_ADD** and **ADAPTER_DEL** messages as part of its event data. It will pass the concerned **INTERFACE_NAME**. Also, as it is a cluster event producer it will additionally pass **NODE_NUMBER**, **NODE_ID** and **CLUSTER_ID** information.

Acceptable monitor files

To monitor for changes in the list of nodes, a monitor file should be created under the **networkAdapterState.monFactory** directory. The monitor file name

```
/aha/cluster/networkAdapterState.monFactory/networkAdapterStateEvent.mon
```

has to be used. No other monitor files may be created in this directory.

Example event data

The following is event data from a **networkAdapterState** event with the default **INFO_LVL**.

```
BEGIN_EVENT_INFO
TIME_tvsec=1271920539
TIME_tvnsec=399378269
SEQUENCE_NUM=1
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=ADAPTER_UP
INTERFACE_NAME=en0
NODE_NUMBER=2
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

clDiskState

The **clDiskState** event producer monitors cluster disks.

Overview

The **clDiskState** event producer resides under the disk directory and monitors cluster disks. This event producer is available only when the system is part of a cluster. This event is generated when a cluster disk goes down or comes up.

Capabilities

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

Return codes

The **clDiskState** returns 0 as the return code. Only if the cluster is removed**AHAFS_CLUSTER_REMOVE (-1)** is returned.

Event producer message

This event producer passes the **DISK_UP** and **DISK_DOWN** messages as part of its event data in the **EVENT_TYPE** field along with the concerned cluster disk name. Also, as it is a cluster event producer it will additionally pass the **NODE_NUMBER**, **NODE_ID** and **CLUSTER_ID** information.

Acceptable monitor files

To monitor cluster disks, a monitor file should be created under the **clDiskState.monFactory** directory. The monitor file name

```
/aha/disk/clDiskState.monFactory/clDiskStateEvent.mon
```

has to be used. No other monitor files may be created in this directory.

Example event data

The following is event data from a **clDiskState** event with the default **INFO_LVL**.

```
BEGIN_EVENT_INFO
TIME_tvsec=1271935734
TIME_tvnsec=265210314
SEQUENCE_NUM=1
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=DISK_DOWN
DISK_NAME=cldisk1
NODE_NUMBER=2
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

repDiskState

The **repDiskState** event producer monitors for repository disks.

Overview

The **repDiskState** event producer resides under the disk directory and monitors for repository disk. This event producer is available only when the system is part of a cluster. This event is generated when a repository disk goes down or comes up.

Capabilities

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

Return codes

The **repDiskState** returns 0 as the return code. Only if the cluster is removed then **AHAFS_CLUSTER_REMOVE (-1)** is returned.

Event producer message

This event producer passes **REP_UP** and **REP_DOWN** messages as part of its event data in the **EVENT_TYPE** field, along with the disk name of the concerned repository disk. Also, as since it is a cluster event producer it will additionally pass **NODE_NUMBER**, **NODE_ID** and the **CLUSTER_ID** information.

Acceptable monitor files

To monitor repository disks, a monitor file should be created under the **repDiskState.monFactory** directory. The monitor file name

```
/aha/disk/ repDiskState.monFactory/repDiskStateEvent.mon
```

has to be used. No other monitor files may be created in this directory.

Example event data

The following is event data from a **repDiskState** event with the default **INFO_LVL**.

```
BEGIN_EVENT_INFO
TIME_tvsec=1271933757
TIME_tvnsec=134003703
SEQUENCE_NUM=1
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=REP_UP
```

```
DISK_NAME=hdisk2
NODE_NUMBER=2
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

diskState

The **diskstate** event producer monitors for local disk changes.

Overview

The **diskState** event producer resides under the disk directory and monitors for local disk changes. This event producer is available only when the system is part of a cluster. This event is generated when a local disk goes down or comes up. This event will be notified only for disks that are supported by the storage framework.

Capabilities

```
AHAFS_THRESHOLD_STATE
AHAFS_CALLBACK_INTRCNTX
```

Return codes

The **diskState** returns 0 as the return code. The **AHAFS_CLUSTER_REMOVE** (-1) is returned only if the cluster is removed.

Event producer message

This event producer passes **LOCAL_UP** and **LOCAL_DOWN** messages along with the concerned local disk name as part of its event data. Also, as a cluster event producer it will additionally pass **NODE_NUMBER**, **NODE_ID** and **CLUSTER_ID** information.

Acceptable monitor files

To monitor local disks, a monitor file should be created under the **diskState.monFactory** directory. The monitor file name of the format

```
/aha/disk/diskState.monFactory/<hdiskn>.mon
```

must be used, with the name of the local disk that has to be monitored.

Example event data

The following is event data from a **diskState** event with the default **INFO_LVL**.

```
BEGIN_EVENT_INFO
TIME_tvsec=1271935029
TIME_tvnsec=958362343
SEQUENCE_NUM=1
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=LOCAL_UP
DISK_NAME=hdisk4
NODE_NUMBER=2
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

vgState

The **vgstate** event producer can verify the status of the VG on a disk.

Overview

The **vgState** event producer resides under the disk directory. This event producer is available only when the system is part of a cluster. Whenever a local (registered with **diskState**) or cluster disk up or down event happens a corresponding **VG_UP** and **VG_DOWN** event is triggered for the volume group residing on that disk. Using this event producer, an application can verify the status of a VG on the disk, with the LVM subsystem.

Capabilities

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

Return codes

The **vgState** returns 0 as the return code. The **AHAFS_CLUSTER_REMOVE** (-1) is returned only if the cluster is removed.

Event producer message

This event producer passes **VG_UP** and **VG_DOWN** messages, as part of its event data. It will pass the concerned disk name and volume group name. Also, as this is a cluster event producer it will additionally pass **NODE_NUMBER**, **NODE_ID** and **CLUSTER_ID** information.

Acceptable monitor files

To monitor for changes in the list of nodes, a monitor file should be created under the **vgState.monFactory** directory. The monitor file name

```
/aha/disk/vgState.monFactory/vgStateEvent.mon
```

has to be used. No other monitor files may be created in this directory.

Example event data

The following is event data from a **vgstate** event with the default **INFO_LVL**.

```
BEGIN_EVENT_INFO
TIME_tvsec=1271915408
TIME_tvnsec=699408296
SEQUENCE_NUM=0
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=VG_UP
DISK_NAME=hdisk3
VG_NAME=myvg
NODE_NUMBER=2
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

Notices

This information was developed for products and services offered in the US.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (your company name) (year).

Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as the customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

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

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

Index

Special Characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z