ARIX CORPORATION

ARIX-OS V.3 Release 2.2
Release Notes

Part Number MA-04618-XX
Revision K

Release Notes
MA-04618-XX, Rev. K

October 1989

This document contains the latest information available at the time of preparation. Therefore, it may contain descriptions of functions not implemented at manual distribution time. To ensure that you have the latest information regarding levels of implementation and functional availability, please consult the appropriate release documentation or contact your local ARIX representative.

ARIX reserves the right to modify or revise the content of this document. No contractual obligation by ARIX regarding level, scope, or timing of functional implementation is either expressed or implied in this document. It is further understood that in consideration of the receipt or purchase of this document, the recipient or purchaser agrees not to reproduce or copy it by any means whatsoever, not to permit such action by others, for any purpose without prior written permission from ARIX.

Table of Contents

Preface

The *Release Notes* contain important information about ARIX Operating System (ARIX-OS) V.3 Release 2.0 on ARIX Computers. They briefly describe the new features of this release, provide installation information, software notes, compatibility notes, and list the documents that pertain to the ARIX-OS.

For a complete description of the procedures used in the administration of an ARIX Computer running ARIX-OS V.3, see the *System Administrator's Guide*.

Additional Products

Remote File Sharing is offered as a product separate from ARIX-OS V.3. Also network transport services required by RFS are provided by a separate product. For a description of software installation and build procedures, software notes, and information about documentation for these products, see the *Remote File Sharing Release Notes* and the release notes for the appropriate network product.

Additional Documentation

These release notes are shipped with every order of ARIX-OS V.3.

There are nine other volumes that can be ordered in sets (full, standard. or optional), or individually. The **Full** set contains all nine volumes. The **Standard** set consists of five volumes and the **Optional** set contains four volumes, as follows:

Standard Set	**Optional Set**
Assist User's Guide MA-99255-10	Network Programmer's Guide MA-99255-01
System Administrator's Guide MA-99255-06	Programmer's Guide MA-99255-02
System Administrator's Reference Manual MA-99255-07	Programmer's Reference Manual MA-99255-03
User's Guide MA-99255-08	STREAMS Programmer's Guide MA-99255-05
User's Reference Manual MA-99255-09	

For more information about these volumes, contact an ARIX representative.

Conventions Used in These Release Notes

In this document certain typesetting conventions are followed when command names, command line formats, files, and directory names are described. There are also conventions for displays of terminal input and output.

- Enter words that are in the **bold** font as they appear.
- All text entered by the user on the command line must end with a Carriage Return.
- Sometimes commands require the Control <Ctrl> key to be held down while pressing another key. For example, holding the Control key and pressing the d key is shown in text as Ctrl-D.
- *Italic* words are variables; substitute the appropriate values. These values may be file names or they may be data values, as applicable.
- CRT or terminal output and examples of source code are presented in `constant-width` font.
- Characters or words in square brackets, [ ], are optional. (Do not type the brackets.)

A command name followed by a number, for example, **ed**(1), refers to that command's manual page, where the number refers to the section of the manual. Manual pages from section (1) appear in the *User's Reference Manual*, unless otherwise noted. Manual pages from sections (2) and (3) appear in the *Programmer's Reference Manual*. Manual pages from section (1M), (4), (5), and (7) appear in the *System Administrator's Reference Manual*.

Examples in these *Release Notes* show the default system prompt for ARIX-OS V.3, the dollar sign (**$**). They also show the default prompt when logged in as the superuser, the pound sign (#).

Features of ARIX-OS V.3 Release 2.0

Release 2.0 maintains source code compatibility with previous ARIX releases (with the exceptions noted below), and object code compatibility at the application level. Application packages, such as DOCUMENTER'S WORKBENCH Software continue to work with Release 2.0.

ARIX-OS V.3 Release 2.0 provides the following new features:

- Internationalization
  - Support for 8-bit Code Sets
  - Support for Alternate Date and Time Formats (see **cftime**(4) and **environ**(5) in the *System Administrator's Reference Manual*)
  - Support for Alternate Character Classification and Conversion Rules (see **chrtbl**(1M) in the *System Administrator's Reference Manual*)
- Remote File Sharing (optional product)
- Network File System (optional product)
- Networking Support Utilities
- Enhanced Basic Networking Utilities
- Shared Libraries
- Signal Mechanism Enhancements
- Improved Terminal Support Facilities
  - Terminal Information Utilities Enhancements
- ASSIST Menu/Forms Interface
- New **awk** (**nawk**)
- Device Drivers for the Small Computer Systems Interface (SCSI) board
- Performance Improvements
  - Paged Text Files
  - Faster and Smaller **curses**
  - Faster Archival Routines

- Additional Features
  - New System Header Files
  - Remote File Sharing Loop-Back (optional)
  - Improved Recovery of Files from **cpio** Archives

Support for 8-bit Code Sets

The **cat**, **ed**, **egrep**, **expr**, **find**, **grep**, **ls**, **pg**, **sed**, **sort**, and **vi** commands and the **curses** library were changed so they no longer use the eighth bit of each byte. This change enables these commands to handle code sets where all 8 bits are used. (**nawk** provides this capability for **awk** users.) Because ASCII only uses 7 of the available 8 bits in a byte, some commands made special use of this eighth bit; other commands assumed that if the bit was set, the byte was invalid. Other commonly-used commands, such as **sh** (the shell), already support 8-bit code sets.

In addition, 8-bit characters can be sent between a terminal and the system (see the **init**(1M) and **gettydefs**(4) manual pages in the *System Administrator's Reference Manual*).

Support for Alternate Date and Time Formats

The **cpio**, **date**, **ls**, **mount**, **pr**, and **sort** commands were changed to provide the date and time in the language and format determined by the value of the **LANGUAGE** environment variable. While the United States conventions remain the default, other languages can be supported by creating and installing a file for the language desired in the **/lib/cftime** directory. The content of that file includes: month and weekday names (full and abbreviated), default local time, date, pre-noon, and post-noon formats, and the default output of the **date** command if the **CFTIME** environment variable is not set. In addition, time zones and alternate time zones (such as daylight time) now can be defined in terms of hours and minutes using the **TZ** environment variable.

For more information, see the **date**(1) manual page in the *User's Reference Manual* and the **timezone**(4) and **environ**(5) manual pages in the *System Administrator's Reference Manual.*

Support for Alternate Character Classification and Conversion Rules

cat, **ed**, **egrep**, **grep**, **ls**, **pg**, **sed**, **sort**, and **vi** (commands that convert characters from upper to lower case or classify characters as alphabetic, printable, upper or lower case, etc.), were changed to support code sets or classification rules according to the value of the **CHRCLASS** environment variable. While ASCII remains the default for these operations, other conversion and classification rules are supported by creating and installing a file describing these rules in the **/lib/chrclass** directory. A new administrative command, **chrtbl**, is used to create this file.

For more information, see the **chrtbl**(1M) and **environ**(5) manual pages in the *System Administrator's Reference Manual*.

Remote File Sharing

To take advantage of Remote File Sharing, the optional Remote File Sharing utilities and a transport provider must already be installed. (See "Networking Support Utilities" for a description of a transport provider.)

Remote File Sharing allows users to share files, directories, devices and named pipes transparently among computers that are linked by a network. Files are shared transparently by mounting a remote directory as one would mount a file system. Each computer on the network controls which local resources are available to other computers and which remote resources local users may access. For example, with Remote File Sharing users may share a directory among several departments of business, or a letter-quality printer or typesetter that no one department could support by itself. For more information, see the *Remote File Sharing Release Notes* and the *System Administrator's Guide*.

Network File System

The Network File System is an optional overlay tape for ARIX-OS V.3. To ARIX-OS V.3, it adds the functionality of Sun Microsystem's 3.2 NFS release. The Network File System is a facility for sharing files in a heterogeneous environment of machines, operating systems and networks. With NFS, users can mount directories across the network and then treat remote files as if they were local.

Networking Support Utilities

The Networking Support Utilities provide STREAMS tools, the AT&T Transport Interface, and the network Listener. STREAMS is a set of tools for development of communication and networking services within ARIX-OS V.3; the Transport Interface is based on the Transport Service Definition (Level 4) of the International Organization for Standardization (ISO) Open Systems Interconnect (OSI) reference model, and defines how a user accesses the services of a transport protocol; the Listener receives requests for network services from other systems, interprets which service is needed, and starts a process that has been named to provide the requested network service. The listener then drops out of the communications path and continues to listen for new service requests.

For more information about the Networking Support Utilities, see the *System Adminstrator's Guide* and the *STREAMS Programmer's Guide*.

Enhanced Basic Networking Commands

Basic networking commands (for example **uucp**(1C) and **uux**(1C)) have been enhanced to conform to the AT&T Transport Interface. These utilities can communicate using any Transport Provider that conforms to the AT&T Transport Interface.

Users can show which, if any, Transport Providers are available with **nlsadmin -x**. If additional Transport Providers are installed, no changes are needed to the software to accommodate the underlying media or protocols; simply register Basic Networking Utilities services with the network Listener for those Transport Providers and register the Transport Providers in the administrative files for the Basic Networking Utilities (see **nlsadmin**(1M) in the *System Administrator's Reference Manual*).

Even with these enhancements the syntax of the basic networking commands is no different from before. See the *System Administrator's Guide* for details about how to manage this facility.

Shared Libraries

The C programming utilities now allow the programmer to build a shared library of routines accessed at run-time, rather than having those routines combined with an application program at load time. For more information about generating shared libraries see the *Programmer's Guide*.

Signal Mechanism Enhancements

A new set of system calls (see the **sigset**(2) manual page in the *Programmer's Reference Manual*) provides a mechanism to catch and hold signals without losing them during later processing, and to guarantee that a process reaches the signal handler before it is interrupted by another signal. Some additional signal-handling features, provided by other popular operating systems, are also available.

ASSIST Utilities

The ASSIST Utilities package provides on-line assistance for ARIX-OS V.3 users. It is designed to help new users get started using the system and to help experienced users explore ARIX-OS V.3 system facilities that lie outside their areas of expertise. Rather than hiding the ARIX-OS V.3 system, ASSIST exposes and explains it. **assist** does this through four components:

- Menus: show categories of activities initiated by a user on a computer and list ARIX-OS V.3 system commands for each of these activities.
- Command Forms: construct an ARIX-OS V.3 system command line at the bottom of the screen resulting from a user's answer to questions about what they want to do; then, optionally, execute the command line.
- Keyword Search: suggest appropriate ARIX-OS V.3 system commands based on key words typed by the user.

- Walkthrus: demonstrate, interactively, important concepts underlying the ARIX-OS V.3 system and commands that have their own user interfaces, including how to use ASSIST.

In addition to these four components, the **astgen** commands allows users to add to or modify menus and command forms.

For more information, see the *ASSIST Software User's Guide* and the **assist**(1) and **astgen**(1) manual pages in the *User's Reference Manual.*

New awk (nawk)

awk is a programming language for information retrieval and data manipulation that is often used by people with no programming background. Customers using **awk** in an international environment must use the new version (**nawk**), because the old version does not support 8-bit code sets. Some of the other enhancements to this version of **awk** include:

- the ability to define functions
- new keywords: **delete**, **do**, **func**, **function**, **return**
- new built-in functions: **atan2**, **cos**, **sin**, **rand**, **srand**, **gsub**, **sub**, **match**, **close**, and **system**
- new pre-defined variables: **FNR**, **ARGC**, **ARGV**, **RSTART**, and **RLENGTH**
- the input field-separator variable, **FS**, and the third argument to **split** are treated as regular expressions
- the precedence of operations now matches C language precedence

Because some of these enhancements may not be compatible with some existing **awk** programs, users will get the old version when **awk** is entered. To take advantage of these new features, use the **nawk** command.

SCSI Controller Boards

Small Computer Systems Interface (SCSI) device drivers have been added to the kernel to support the ARIX dual-channel SCSI controller board. The SCSI board is a full-function controller capable of operating alone or in combination with any other ARIX Disk/Tape controller (EDT or HSDT) board. A maximum of four SCSI boards can be configured on the ARIX system. Each board can support 14 SCSI devices. Each disk drive on the SCSI controller can be subdivided into 16 logical disks (or slices).

Paged Text Files

A new **a.out** file type has been created to take advantage of ARIX-OS V.3 paging. Text files of this type are paged in on demand instead of being loaded in their entirety before execution. See **ld**(1) in the *Programmer's Reference Manual.*

Smaller and Faster curses

In many cases, by entirely recompiling and relinking an application package with the Release 2.0 **curses** library, the resulting application may use significantly less memory than it did in earlier releases, and may execute much faster. If object modules from earlier releases are used in conjunction with the Release 2.0 **curses** library, performance improvements will not be as great because of the code and space required to maintain object-code compatibility.

For more information about **curses**, see the "Software Notes" section of the *Release Notes* and the **curses**(3X) manual page in the *Programmer's Reference Manual.*

Faster Archival Routines

The **cpio** utility has been enhanced to provide faster output to the 150MB tape drive.

A new utility, **ftar**, has been added to /usr/bin which provides faster output to the 150MB tape drive.

New System Header Files

A new header file has been added to **/usr/include**. **limits.h** contains definitions for commonly used values that vary between implementations. Several new definitions were added to the header file **/usr/include/sys/stat.h** to make it easier for programmers to write portable code.

Remote File Sharing Loop Back

The loopback feature enables users to simulate the higher levels of Remote File Sharing (RFS) functionality within one ARIX Computer. Application programs designed to use RFS can be tested partially without actually communicating with a remote computer. This feature can also be used to demonstrate RFS when only one computer is available.

This feature is provided by Release 1.0 of the optional Remote File Sharing Utilities. For information describing how to use this feature, see the **–o** option on the **rfadmin**(1M) manual page in the *System Administrator's Reference Manual.*

Improved Recovery of Files from cpio Archives

If errors are encountered while restoring a file using **sysadm** or **cpio**, users can now skip over the bad blocks and continue the restore with the next file. To invoke this procedure, select the **restore** command from the **sysadm filemgmt** menu or use the **–k** option described on the **cpio**(1) manual page in the *User's Reference Manual.*

Prerequisites of ARIX V.3 Release 2.0

To successfully load and run ARIX-OS V.3 Release 2.0, systems containing any of the boards listed below must be at or above the listed revision level:

BOARD TYPE	NAME	PART NUMBER	REVISION LEVEL
CPU 25MHz	CPU32	80-02204-02A	B4 [1]
CPU	CPU32	80-01428-02	Bd [1]
CPU 25MHz	CPU32	80-02204-02B	B3 [2]
CPU	CPU32	80-01428-02	Bb [2]
4MB Memory	2/8 MB MEM	80-00577-06	B6
8MB Memory	2/8 MB MEM	80-00577-08	B6
8MB Memory	8/16/32 MB MEM	80-02735-02A	A1
16MB Memory	8/16/32 MB MEM	80-02735-12A	A1
32MB Memory	8/16/32 MB MEM	80-02735-22A	A1
I/O	HSDT	80-00580-02	Ah
I/O	EDT	80-01676-02	B7
I/O	SCSI	80-03538-02	A
I/O	EGC	80-01924-02	A4
I/O	GC-8	80-00653-02	B5
I/O	GC-16	80-02206-02	A
I/O	MAC	80-01127-02	B
DMC	DMC-4	80-01181-02	Ac
DMC	DMC-4/2	80-01181-12	Ac

[1] Lowest revision acceptable for use with SCSI boards.

[2] Lowest revision acceptable for use with EDT or HSDT boards.

In order to successfully install this product, superuser privileges must be used.

The kernel will be reconfigured when installing the software, so it is imperative to have some experience in reconfiguring UNIX kernels.

The ARIX–OS V.3 Release 2.0 bootimage and kernel, as delivered on the release tape, must run together. Older bootimage files will not run with this kernel.

Installation Procedures

This section provides installation procedures to install Release 2.0 of the ARIX Operating System (ARIX-OS) V.3 on ARIX systems.

ARIX-OS V.3 consists of an International and a Domestic release and supports ONLY 68020 CPU processing.

Please DO NOT attempt to upgrade a system that contains 68000 CPUs with this release.

This section provides procedures for loading the software on a **new system** and for upgrading an **existing system**. The procedures should be followed carefully to ensure proper installation of ARIX-OS V.3.

If you are upgrading your system from ARIX-OS V.3 Release 1.X, start with the subsection "Upgrading an Existing System" and follow the procedure.

If you are loading ARIX-OS V.3 for the first time, skip to the subsection "Loading a New System" to install the system software.

ARIX-OS V.3 is distributed on 1/4 inch cartridge or 9-track tape. The distribution tapes for the release are as follows:

- Bootimage, Root, and Usr (root and /usr file systems)
- Domestic Overlay (utilities using encryption algorithms for domestic sites)

This section should be read completely before loading the distribution tape.

The default names for the cartridge tape drive are **/dev/rmt1** and **/dev/rmt0** for rewind and no-rewind respectively. The default names for the 9-track tape drive are **/dev/9mt1** and **/dev/9mt0** for rewind and no-rewind respectively.

NOTE

Throughout these procedures, it is assumed that a cartridge tape device is being used. If this is not the case, enter the appropriate device names where applicable.

Upgrading an Existing System

To upgrade an existing system, specially configured files need to be backed up prior to loading ARIX-OS V.3 Release 2.0. A blank tape as well as the distribution tapes are required.

System Shutdown

Whenever the system must be shut down, such as for backups or a reboot, the program **/etc/shutdown** should be run while logged in as superuser. The shutdown procedure is designed to perform an orderly system shut down, terminating all processes and bringing the system back to single-user state with all buffers flushed.

The interval of time between sending a warning message out and actually shutting down the system can be specified. This time period is the number of seconds of delay. For example, specifying a period of 300 will result in a 5-minute delay before the actual shutdown. The system has a default period of 1 minute.

A personalized message can also be sent if desired. A default message is sent to all logged-in users if a personalized message is not sent.

To initiate a typical shutdown, perform the following procedure:

1. Log in as superuser.
2. Change to the root (/) directory (if not already there), then enter the shutdown command:

   cd /
   /etc/shutdown

   In the above example, the default is being used which only gives the users 60 seconds (1 minute) to write and close their files before system processes are killed.

The system will respond with the following:

```
SHUTDOWN PROGRAM
Fri Jul 31 16:51:58 EST 1987
Do you want to send your own message (y or n)?
```

3. To send a personalized message, type **y**, and the message followed by a **Ctrl-d**.

```
System maintenance about to begin.
Please log off.
```

The system will now wait for the number of seconds specified, or for the default of 60 seconds.

```
SYSTEM BEING BROUGHT DOWN NOW!!!
Busy out (push down) the appropriate phone lines for this system.
Do you want to continue (y or n)?
```

4. To quit the shutdown program, enter **n**, and the multi-user prompt will appear. To continue the shutdown program, type **y**, and the system prompts with the following.

```
Process accounting stopped
Error logging stopped
All currently running process will now be killed
(Data may appear here if processes are being killed.)
Wait for "INIT: SINGLE-USER MODE" before halting
#
INIT: New run level: S

INIT: SINGLE-USER MODE
#
```

NOTE

If any changes are made to files from single user state, the **sync** command should be issued to ensure that the changes are flushed from main memory back to disk (this occurs automatically in multi-user state).

5. To verify the status of the file systems after the **shutdown** procedure has been performed, enter the following:

   mount

   This command displays any file systems that are still mounted. If any file systems except root **are** mounted, they will have to be manually unmounted using the **umount** command.

6. To flush the buffer from the changes that were made above, enter the following:

   sync
   sync

Saving Site Dependent Files

To upgrade a system to ARIX-OS V.3 Release 2.0, files that are specially configured for the local system must be backed up prior to loading the new software release tape. The backed up files should then be returned to the system once the new software is loaded.

1. If not already done, bring the system into single-user state by running the **shutdown** program (described in the previous subsection).

2. Remount the /usr file system:

   mount /dev/dsk/c0d0s2 /usr

   The system will respond with the following message:

```
mount: warning! <> mounted as </usr>
#
```

3. Executing the following commands creates a file (**newonly**) with a list of files that were not on the original release tape.

   cd /local/bin
   find / -depth -print | sort > newlist
   sort /etc/syslist > oldlist
   diff oldlist newlist | grep '^>' | sed 's/^> //' > newonly

   Note that there should be one space between the '>' and the '/' in the **sed** search statement.

4. The system administrator should edit the file **newonly** and remove any file names that shouldn't be moved to the upgraded system (i.e. junk files in /tmp).

NOTE

Only the HoneyDanBer version of **uucp**(1) is supported with ARIX–OS V.3 Release 2.0. Sites not using the HoneyDanBer version of **uucp** should refer to the *System Administrator's Guide* for implementation details.

5. The file **/local/bin/savelist** contains the basic list of site dependent files. These are the files that will be backed up by default. The list of site dependent files contained in the file **newonly** should then be combined with **savefiles** by using the following command:

```
cat newonly savelist > newsavelist
```

6. The V.3 functionally improved backup script, **/local/bin/savefiles**, is located on the release tape labeled "Bootimage, Root, and Usr." Load this tape onto the tape drive.

7. Read past the bootimage portion using the following command:

```
< /dev/rmt0
```

8. Load the backup script from tape:

```
cd /
cpio -idvBmu "/local/bin/savefiles" < /dev/rmt0
```

9. Load an empty tape onto the tape drive on the system. This tape will be used for the backups.

10. The backup script uses **cpio**(1) to make a copy of the files. Enter the following to perform the backup:

```
cd /
/local/bin/savefiles /local/bin/newsavelist /dev/rmt1
```

The files will be loaded onto the tape with a hyphen (-) as the last character in the name.

11. Confirm that the tape was written properly with the command:

```
cpio -itvB < /dev/rmt1
```

12. Remove the tape from the tape drive.

NOTE

The site dependent files have now been backed up. The procedure for loading the system software on an upgrade system is the same as for loading a new system. To continue the installation procedure of ARIX–OS V.3 Release 2.0, follow and complete all procedures in this section this section.

Loading a New System

This subsection contains instructions to load ARIX-OS V.3 on a new system.

Systems are shipped from ARIX with the disks formatted and set up. It is possible that during shipment the setup may have been distroyed.

If the disk is not formatted and/or set up, refer to the "Disk/Tape Management" section of the *System Administrator's Guide* to do so.

Loading Bootimage

Perform the following procedure to load the Bootimage program onto the system.

1. For software installation purposes, the "Auto Start" switch should be turned off by performing the following:

   For a Model 825/800, flip the "Auto Start" switch located at the lower rear of the system to the OFF position.

   For all other models, open the front door, and on the system status panel flip the "Auto Start" switch to the OFF position.

2. Power on the system.

3. Reset the system:

   For a Model 825/800, at the front of the system, turn the key to RESET then ON.

   For all other systems, open the front door, and on the system status panel press the red RESET button.

The system will then cycle through the following test sequences:

```
initializing memory
testing edac circuitry
testing memory
```

When the system is done testing, text similar to the following appears:

```
9600 baud

CPU32 MONITOR ver v1.X
power-up sequence
memory edac enabled
totram XXXXXX
master cpu - slot XX
>
```

BOOTIMAGE OVERLAY 2.2

4. Insert the tape labeled "~~Bootimage, Boot, and Usr~~" into the tape drive.
5. From the console terminal at the (>) prompt, enter the following command to boot the system (press the space bar in place of <space>):

   b<space>

   A message similar to the following appears:

```
Boot (VER 5.xx)
Loading EDT code (VER 1.xx)
x-1000 Boot
:
```

6. Enter the following command at the standalone (:) prompt to load the standalone program:

   ldsa

   The system prompts with the following:

```
Standalone ldsa (VER 5.xx)
Source device <CR> defaults to TAPE device c0d0:
```

7. Press the **Carriage Return** after the above prompt. As a result, the system displays the following:

```
Target device <CR> defaults to DISK device c0d0r:
```

NOTE

Copying the bootimage on all physical drives may be desired, so that booting is possible from any drive. To do so, specify the disk drive as **cXdYr** instead of <CR> (where X is the controller number and Y is the drive number).

In the previous command, "c0" refers to controller 0 and "d0" refers to physical drive number 0.

8. Enter a **Carriage Return** after the prompt for the default. The system prompts with text similar to the following:

```
Copying BOOTIMAGE to c0d0r.
xxxxx bytes transferred

(0) returned by exit
x-1000 Boot
:
```

The "xxxxx bytes transferred" message indicates that the loading of the Standalone Program is complete.

9. To verify that the Standalone Program was properly loaded onto the system, the system should be reset as follows:

   For a Model 825/800, at the front of the system, turn the key to the RESET then to the ON position.

   For all other systems, open the front door and on the system panel, press the red RESET button.

   The system then cycles through the following test sequences:

```
initializing memory
testing edac circuitry
testing memory
```

When the system is done testing, text similar to the following appears (the display will differ if your system has a 25MHz CPU or a CPU that supports SCSI controllers):

```
9600 baud

CPU32 MONITOR ver v1.X
power-up sequence
memory edac enabled
totram XXXXXX
master cpu - slot XX
>
```

10. After the (>) prompt, type the following (press the space bar in place of <space>):

    b<space>

The system displays text similar to the following:

```
Boot (VER 5.xx)
Loading EDT code (VER 1.xx)
x-1000 Boot
:
```

INSERT BOOTimake, ROOT & RISK TAPE 2.0

Loading the Root File System

Perform the following procedure to make and load the root file system.

1. Type the following command at the standalone (:) prompt:

   mkfs

   The system displays text similar to the following:

```
Standalone mkfs VER 5.xx

Select one
[m]ake a blank file system:   or
[r]ecreate a file system from tape:
```

2. Enter the following command to recreate a file system from tape:

   r

   The system prompts with the following:

```
Load the file system tape.
Enter the tape device number.
<cr> defaults to c0d0:
```

3. Verify that the tape labeled "Bootimage, Root, and Usr" is in the tape drive, then enter a **Carriage Return**. As a result, the system displays the following:

```
Does this tape contain both a Bootimage and a Root File System?
```

4. Since the release tapes include both the Bootimage and Root file systems, answer with a **y** followed by a **Carriage Return**. The system then prompts with the following:

```
file system size:
```

5. Enter a **Carriage Return** to use the file system size that was made in the **dsetup** program. As a result, the system displays the following:

```
file system:
```

6. Enter the following:

c0d0s0

In the previous command, "c0" refers to controller 0, "d0" refers to physical drive number 0, and "s0" refers to slice or logical disk number 0.

NOTE

To change the installation of the root file system to another location, substitute the correct ids of the controller, drive and slice for those used above

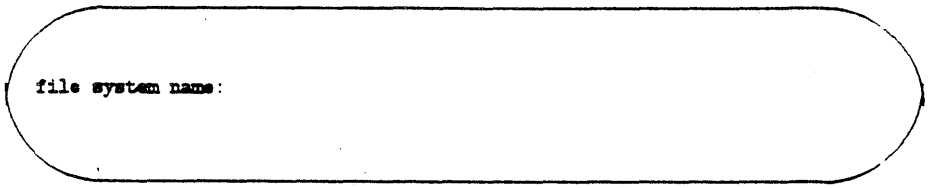
The system displays the following:

```
file system: c0d0s0
fsize=xxxxx
isize=xxxx
interlace x, sectors/cylinder xxx
Verbose mode?
```

7. Enter either a **y**, or a **n** followed by a **Carriage Return** depending on whether verbose mode is wanted or not.

If verbose mode is chosen, the names of the files on the root file system scroll on the screen as they are loaded to the disk. If verbose is not chosen, the screen remains as is until all files are loaded to the disk. This process takes approximately 15 minutes.

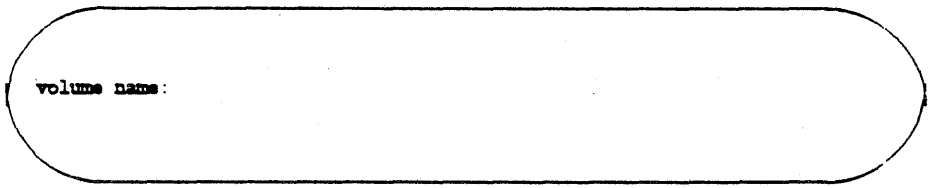
After the file system has been loaded from tape, the following message is displayed:



NOTE

An optional 6 character name can be input here if desired. Refer to the **labelit**(1M) command in the *System Administrator's Reference Manual* for details.

8. Enter a **Carriage Return** after the above prompt. The system displays the following:



NOTE

An optional 6 character name can be input here if desired. Refer to the **labelit(1M)** command in the *System Administrator's Reference Manual* for details.

9. Enter a **Carriage Return**. As a result, the tape rewinds and the system displays the following:

```
(0) returned by exit
x-1000 Boot
:
```

10. After the root file system is loaded, enter a **Carriage Return** at the (:) prompt for the default. Text similar to the following should appear on the screen:

```
: c0d0s0syst

ARIX-OS V.3 Release 2.0 xxx (kernel version v5.xx)
(c) 1988 ARIX Corporation
Node syst
real mem = xxxxxx
avail mem = xxxxxx
x processor system
x floating point processor(s)

INIT: SINGLE USER MODE
#
```

11. At the (#) prompt, enter the following to verify and adjust the disk device nodes to match the hardware configuration:

/etc/disks

12. At the (#) prompt, initialize the /etc/mnttab file with the command:

devnm / | setmnt

13. At the (#) prompt, run **fsck**(1M) to ensure that the root file system has been loaded properly.

    fsck /dev/dsk/c0d0s0

    The **fsck** program checks the file systems for any discrepancies. The system displays information similar to the following:

```
/dev/dsk/c0d0s0
File System:  Volume:

** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
xxx files xxxx blocks XXXXX free
#
```

If there are no errors, the root file system has now been successfully loaded. If there are "Boot No Sync" errors, RESET the system, boot the system, and run **fsck**(1M) on the root (/) file system.

Loading the Usr File System

Perform the following procedure to load the Usr portion of the tape.

1. To make and load the /usr file system, enter the following at the (#) prompt:

   mkfs /dev/rdsk/c0d0s2

   The system prompts with text similar to the following:

```
Mkfs: /dev/rdsk/c0d0s2?
(DEL if wrong)
bytes per logical block = xxxx
total logical blocks = xxxxx
total inodes = xxxxx
gap (physical blocks) = x
cylinder size (physical blocks) = xxx
#
```

After the **mkfs** command is invoked, the program echoes the device name that was entered and pauses. If an incorrect device name was typed in, the program can be interrupted with the delete key <DEL>.

2. Type the following command after the (#) prompt:

   mount /dev/dsk/c0d0s2 /usr

   The system prompts with the following:

```
mount: warning! <> mounted as </usr>
#
```

3. Change the mode, owner and group of the /usr file system with the command:

   chmod 755 /usr
   chown bin /usr
   chgrp bin /usr

4. Verify that the tape labeled "Bootimage, Root, and Usr" is still in the tape drive.

5. If installing a cartridge distribution or if the device nodes supplied with the ARIX Model 875/1600 have been altered, skip to step 6. This step creates the nodes for a 9-track device only. Enter the following command:

   /local/bin/mk9mt

6. Change to the /usr directory by entering the command:

   cd /usr

7. Position the tape at the /usr file system by entering the following commands (the "<" is the less than sign and is followed by the device name):

   < /dev/rmt1
   < /dev/rmt0
   < /dev/rmt0

   After the above rewind and no rewind commands, the tape is positioned in front of the /usr files.

8. When the (#) prompt returns, type the following:

   cpio -idvBmu < /dev/rmt1

   The names of the files on the /usr file system are displayed on the screen as they are loaded to the disk. This process takes about 20 minutes.

9. To verify that the /usr file system was loaded properly, enter the following commands:

   cd /
   umount /dev/dsk/c0d0s2
   fsck /dev/rdsk/c0d0s2

   Text similar to the following should appear on the screen:

```
/dev/dsk/c0d0s2
File System: Volume

** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
xxx files xxxx blocks XXXXX free
#
```

10. If the system contains a CPU32 Revision Bb (or higher) board which has the **Autoboot** option, perform the following:

    For a Model 825/800, flip the "Auto Start" switch located at the lower rear of the system to the ON position.

    For all other models, at the front of the system on the status panel, flip the "Auto Start" switch to the ON position.

    For additional information on **autoboot**, refer to the **init**(1M) manual page in the *System Administrator's Reference Manual* and the **inittab**(4) manual page in the *Programmer's Reference Manual.*

If there are no errors, the /usr file system has been successfully installed. If there are errors, correct the problem and run **fsck**(1M) on the /usr file system.

To continue setting up the system (both new and upgrade systems), follow the procedures in the subsection "Loading the Domestic Overlay Tape."

Loading the Domestic Overlay Tape

For domestic systems, the Domestic Overlay tape can now be loaded by performing the following procedure.

NOTE

For International sites, skip to the subsection "Reconfiguring GC Board Nodes" that follows.

1. When the (#) prompt appears, put the tape in the drive and enter the following commands to extract the information from the tape into the root (/) directory:

   cd /
   mount /dev/dsk/c0d0s2 /usr
   cpio -idvBmu < /dev/rmt1

   The names of the files on the Domestic Overlay tape scroll on the screen as they are being loaded to the disk.

2. Now, unmount the /usr file system using the following command:

   umount /dev/dsk/c0d0s2

3. At the (#) prompt, enter the following to check the root and /usr file system:

   fsck /dev/rdsk/c0d0s0 /dev/rdsk/c0d0s2

   The system should display text similar to the following:

```
/dev/dsk/c0d0s0
File System:  Volume:

** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
xxx files xxxx blocks XXXXX free
#

/dev/dsk/c0d0s2
File System:  Volume:

** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
xxx files xxxx blocks XXXXX free
#
```

The Domestic Overlay tape is now loaded on the system. Both new and upgrading systems should continue with the next subsection to reconfigure the GC board nodes.

Reconfiguring GC Board Nodes

New software is included in Release 2.0 that dynamically allocates the identification number for tty nodes on GC boards.

If new GC-16 boards are being added, they should be inserted to the left of any existing GC boards to avoid any changes to the existing system configuration. For more information on installing and removing logic and interface boards, refer to the section "Printed Circuit Boards" in the *Installation and Maintenance Manual* for your ARIX system.

The tty nodes associated with GC boards will need to be reconfigured under the following circumstances:

- Upgrading from ARIX-OS V.3 Release 1.X
- Upgrading from ARIX-OS 4.0 with GC 3.0 board software
- Exchanging a GC or EGC board for a GC-16 board.

Follow any of the procedures below that may apply to your system upgrade. Refer to the *System V.3 System Administrator's Guide* for greater detail about the following steps.

Upgrading from ARIX-OS V.3 Release 1 or ARIX-OS 4.0 with GC 3.0

1. Remake all tty nodes with **sysadm mktty**(1) so there are no gaps in the tty numbering sequence.
2. Update the file /etc/inittab to reflect the new tty node numbers.
3. Use the **–v** option of **lpadmin** to reconfigure each printer device attached to a GC or EGC board and for each printer device not attached to the first controller board.
4. Reconfigure each modem attached to a GC or EGC board and each modem not attached to the first controller board. Update the **uucp** configuration files to reflect this change.
5. Update any application packages, languages, etc., that maintain device tables and that are affected by the new tty node numbering (i.e., SMC Basic).

6. If new GC-16 boards are added and the total number of tty nodes is greater than 88, modify the line in **/etc/master** that starts with "gctty" by changing the ninth field to **16** (it currently should be 08). Remake the kernel. This change increases the total number of tty nodes from 88 to 176.

Exchanging a GC or ECG Board for a GC-16 Board

1. Use **sysadm mktty**(1) to make the 8 additional tty nodes for each board being exchanged.
2. Use the **–v** option of **lpadmin** to reconfigure each printer device previously attached to a GC or EGC board and for each printer device not attached to the first controller board.
3. Reconfigure each modem previously attached to a GC or EGC board and each modem not attached to the first controller board. Update the **uucp** configuration files to reflect this change.
4. If new GC-16 boards are added and the total number of tty nodes is greater than 88, modify the line in **/etc/master** that starts with "gctty" by changing the ninth field to **16** (it currently should be 08). Remake the kernel. This change increases the total number of tty nodes from 88 to 176.

Both new and upgrading systems should continue with the next subsection "Editing /etc/brc.d/S30ldicb."

Editing /etc/brc.d/S30ldicb

This section describes the procedure to configure a setup file (/etc/brc.d/S30ldicb) that automatically downloads GC driver software to the controller boards when the system is started up.

The command **/etc/ldicb** (that's an el, not a one) downloads the GC driver software (tty.gcp8 or tty.gcp16) to the appropriate controller board.

When the script **/local/bin/gcttyconf** is run, it creates the file **/local/bin/brctty**, which contains the **/etc/ldicb** commands for each controller board contained in the system. For example, if a system has 3 GCs with a GC-16 between the second and third GC, the **/local/bin/brctty** file would contain the lines:

```
/etc/ldicb -g0 /etc/tty.gcp8
/etc/ldicb -g1 /etc/tty.gcp8
/etc/ldicb -g2 /etc/tty.gcp16
/etc/ldicb -g3 /etc/tty.gcp8
```

The syntax is "**ldicb -g**[#] *filename*" where the optional # indicates the GC controller to load (counting from 0). If the controller # is absent, **ldicb** loads all GC cards with the same code (filename). The tty code for the GC-16 card is /etc/tty.gcp16. The tty code for the GC and EGC card is /etc/tty.gcp8. (The file tty.gcp, from previous ARIX releases, has been replaced with tty.gcp8 and tty.gcp16.)

These lines then need to be transferred to the startup file **/etc/brc.d/S30ldicb** (el, not one).

The procedure to set up the controller download code follows:

1. Run **/local/bin/gcttyconf** to create the file **/local/bin/brctty**.

   /local/bin/gcttyconf

2. Edit the file /etc/brc.d/S30ldicb with the editor of your choice:

   cd /etc/brc.d
   vi S30ldicb

3. Search for the **/etc/ldicb** command line(s) and replace them with those just created in the /local/bin/brctty file.
4. Write the changes that were just made to the file **S30ldicb**.

The script to download GC board code is complete. New and upgrading systems should continue with the system installation by creating the lost+found directories as described in the next subsection.

Creating lost+found Directories

Each mountable file system should have a "lost+found" directory. The **fsck** utility copies files to this directory when it cannot place them. For each mountable file system, the system administrator should create an appropriately sized lost+found directory in the top directory of each mounted file system. The ARIX-OS V.3 Release 2.0 tape contains lost+found directories for both the root and /usr file systems.

The script **/local/bin/lost+found** sets up the lost+found directory that should be present on each file system. The script should be run after making a new file system.

If you are loading a new system, skip to the subsection "Multi-User State" to finish the system installation. Systems that are being upgraded should follow the procedure in the next subsection for restoring site dependent files.

Restoring Site Dependent Files

If your system is a first time installation, skip to the next subsection "Multi–User State" and follow the procedure. Continue with this subsection only if you are upgrading your system and have saved site dependent files.

The site dependent files that were backed up at the beginning of this installation procedure can now be restored onto the newly configured system

The restoration should be initiated from single user mode.

1. Change to the root file system with the command:

   cd /

2. Check that the /usr file system is mounted:

   mount

3. If /usr is not mounted, execute the following command. If /usr is mounted, skip to the next step.

   mount /dev/dsk/c0d0s2 /usr

4. Place the backup tape of the saved site dependent files in the tape drive, and generate a listing of the saved files with the following command:

   cpio -itvB < /dev/rmt1 > /tmp/sitefiles

   The files that were saved have a dash appended to them (for example, /etc/passwd-) to differentiate them from the newly installed files.

5. Restore the files from the backup tape with the following **cpio** command:

   cpio -iDmuvB < /dev/rmt1

6. The list of files /tmp/sitefiles should be reviewed to determine on a case by case basis which newly installed files should be overwritten by the backed-up files and which should be merged. For example, the **sysadm** login IDs (setup, listen, sysadm, checkfsys, mountfsys, makefsys, umountfsys and powerdown) in the /etc/passwd file should be merged into the /etc/passwd- file.

7. After all files have been evaluated and updated if necessary, move each backup file to the original file with the command:

   mv file- file

   where file is the name of the file you are restoring.

8. The file restoration is now complete. Continue with the next subsection to bring the system to multi-user mode.

Multi-User State

To enter the multi-user state from single-user state, perform the following procedure:

9. At the (#) prompt, type:

   init 2

   As a result the following should appear:

```
INIT: New run level: 2
Is the date xxx xxx x xx:xx:xx xxx 19xx correct? (y or n)
```

10. If answering with **n**, the following appears:

```
Enter the correct date:
```

Respond in the form of 'mmddhhmmyy' where the starting mm represents the number of the month, the dd is the number of the day, the hh is the hour (in 24 hour time), mm is the minutes and yy is the last two digits of the year. All fields **must** be two digit numbers. Type in today's date as follows:

mmddhhmmyy

As a result the following appears:

```
Is the date xxx xxx x xx:xx:xx xxx 19xx correct? (y or n)
```

the date may be corrected here by answering with **n** or accept the date by answering with **y**.

11. When answering with **y**, the following appears:

```
Do you want to check the file systems? (y or n)
```

To check the file systems, enter a **y**, otherwise enter a **n**. If answered with yes, text similar to the following should appear:

```
/dev/dsk/c0d0s2
File System:  Volume:

** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
xxx files xxxx blocks XXXXX free

/dev/dsk/c0d0s0
File System:  Volume:

** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
xxx files xxxx blocks XXXXX free



console login:
```

At this point, the system is completely initialized, loaded, and ready to operate in a multi-user state. If the login is performed, system messages are displayed along with the prompt '$'.

Additional Installation Information

software application packages

Software upgrades will be necessary for kernel dependent software application packages such as the networking and communications packages (i.e., BSC, SNA, RFS, NFS, X.25 and Ethernet). The release of the application package must be compatible with the ARIX-OS V.3 kernel in use.

/etc/TIMEZONE

To make use of the extended timezone functionality for internationalization, the timezone specifications in **/etc/TIMEZONE** must be enclosed in double quotation marks ("). This is necessary because the semi-colons are part of the specification, and when the shell sees them, it interprets them as the end of a command. See **ctime**(3C) in the *Programmer's Reference Manual and* **cftime**(4), **timezone**(4), and **environ**(5) in the *System Administrator's Reference Manual.*

Init Cannot Parse shell Metacharacters

init cannot parse shell metacharacters unless they are preceded by a backslash. If metacharacters are used in the specifications in **/etc/TIMEZONE**, be sure to precede them with a backslash.

Changing the Location of assist and astgen Executables

The default directory containing the **assist** and **astgen** executables (**mecho, mforms, msetup, mscript, msearch, tsetup,** and **tforms**) is **/usr/lib/assist/bin**. This directory can be changed by editing the shell scripts named **assist** and **astgen** in **/usr/bin**. Make sure the new location is in the PATH.

Step 1 Set **ASSISTBIN**=*new_directory_name* in **/usr/bin/assist** and **/usr/bin/astgen**. **/usr/bin/assist** and **/usr/bin/astgen** must have read and execute permission for others.

Step 2 Move the executable files to the new directory. The new directory must be accessible and all files executable by others.

Changing the Location of assist and astgen Datafiles

The default **root** directory containing the directories for ASSIST and **astgen** datafiles (**forms, scripts, search, setup, astgen**) is **/usr/lib/assist/lib**. This directory can be changed by editing the shell scripts named **assist** and **astgen** in **/usr/bin**. Make sure the new location is in the PATH.

Step 1 Set **ASSISTLIB**=*new_directory_name* in **/usr/bin/assist** and **/usr/bin/astgen**. **/usr/bin/assist** and **/usr/lib/assist** must have read and execute permission for others.

Step 2 Move the entire datafiles directory structure from **/usr/lib/assist/lib** to the new **root** directory. The new directory must be accessible and executable by others.

Predefining Users' Terminals

The ASSIST setup component asks each user to check their terminal type. If a system administrator is certain that a particular type of terminal works correctly with ASSIST on the machine, then the terminal checking for that terminal can be bypassed for the ASSIST users on the machine.

1. Edit **/usr/lib/assist/lib/setup/assistrc.**
2. For each terminal type to be predefined, add a line containing entries for the following four variables, separated by a space:
   - terminal name: Specify the type of terminal being used, for example, 5620.
   - standout mode: This is also known as highlighting.
   - working function keys: The function keys F1 through F8 are usually located at the top of the keyboard.
   - alternate graphical character set: This character set provides features that enhance terminal screen displays.

- To represent the standout, function key, and character set variables, use a **1** (one) if the terminal has that function and a **0** (zero) if it does not. For example, the following entry says that terminal type **sg666** works with ASSIST and that the terminal has a standout mode and function keys, but it does not have a graphical character set:

  sg666 1 1 0

Changing the ASSIST Shell Prompt

ASSIST provides the prompt <ASSIST> when a user invokes an ARIX-OS V.3 system shell within ASSIST. (Users can create a shell within ASSIST by selecting the ARIX System Subshell item from the ASSIST pop-up menu or by entering the ASSIST command **Ctrl-E** followed by the ARIX-OS V.3 system command **sh**.) This prompt can be changed to another fixed string:

1. Edit the file **/usr/lib/assist/lib/setup/assistrc**
2. Set **PS1**=*new_prompt_string*. For example, the following entry would make **assist**: the prompt that users would see when they selected the ARIX System Subshell item:

   PS1=assist:

Software Notes

This section offers some additional information about ARIX-OS V.3 Release 2.2. Notes about commands, system calls, and files are listed alphabetically and are organized by the Reference Manual where those commands, system calls, or files appear. For example, the section "User Commands" contains notes about commands that are listed in the *User's Reference Manual.* Any further problems may be resolved through the Technical Support Center according to the terms of the maintenance contract.

User Commands

as(1)

When the "islongsdi" table overflows, the assembler does not generate the text size information correctly. The sections in the object file will not match the header information, and the resulting object will not be properly linked.

as(1)

The assembly code:

```
set     x,0xa0000000
global  x
```

no longer sets aside 0xa0000000 bytes of .bss, instead, "x" is treated as an absolute.

as(1)"

The assembler no longer changes the machine code

```
and.w  &0,%d0
or.w   &0,%d1
```

to immediate mode.

as(1)

A warning is now issued if the instruction size changes.

as(1)

Absolute symbols now are given the correct symbol type.

assist(1)

If using terminals with programmable function keys, users may find that the function keys work with some of the application packages and then not with others. The function keys must be initialized to their default values for ASSIST to use them. These default values are sometimes changed by software application packages that make the function keys work by clearing the function keys' original values and substituting their own values. Some ASSIST users initialize their function keys themselves and do not want ASSIST to clear or change them.

To make the programmable function keys work with ASSIST, the default values for the keys must be re-initialized on the function keys before ASSIST is run. In general, information on the default values for function keys and how to reset those values is described in the user manual for the terminal.

Depending on the terminal, users may be able to use the **tput** command to re-initialize the function keys by executing:

```
tput init
```

If **tput** works for the terminals, then the system administrator has the option of setting ASSIST so that it automatically clears and re-initializes function keys for all users of the system. Some people might dislike this, so it is suggested that everyone involved should be asked before making any changes.

To set **assist** and **astgen** to automatically re-initialize the function keys, the system administrator should add the **tput** instruction(s) to the shell scripts **/usr/bin/assist** and **/usr/bin/astgen**:

1. Edit file **/usr/bin/assist**.
2. Insert the following line at the beginning of the file:

```
tput init
```

3. The file should now read as follows:

```
tput init
#ident   "@(#)setup:assist.sh       1.5"
export ASSISTBIN ASSISTLIB
ASSISTBIN=/usr/lib/assist/bin
ASSISTLIB=/usr/lib/assist/lib
exec $ASSISTBIN/mssetup $*
```

4. Write the file.
5. Edit the file **/usr/bin/astgen**.
6. Insert the following line at the beginning of the file:

   tput init

7. The file should now read as follows:

```
tput init
#ident   "@(#)setup:astgen.sh       1.3"
export ASSISTBIN ASSISTLIB
ASSISTBIN=/usr/lib/assist/bin
ASSISTLIB=/usr/lib/assist/lib
exec $ASSISTBIN/tssetup $*
```

8. Write the file.

NOTE

The `#ident` line in the file will not match the one above.

assist(1)

Horizontal and vertical lines in **assist** and **astgen** are drawn as solid lines if the terminal has a graphical character set and as dashed lines otherwise. If horizontal lines composed of "q's" appear on the ASSIST menus or forms, then ASSIST has been told during the terminal checking procedures that the terminal has a graphical character set, but the terminal either does not have a graphical character set or the graphical character set is not enabled.

Depending on the terminal, the graphical character set may have to be enabled each time the terminal is turned on. The terminal's user manual should explain how to do this. Also, the **tput** sequences described in the section on programmable function keys can be tried.

The command **assist –s** (choice **a**) can be used to re-evaluate the terminals' graphical character set. Modifications have been made to the scripts that build the assist menus so that the menus are built correctly.

astgen(1)

A shell script with %, #, or & in it can cause **astgen** to prematurely exit, causing users to lose all work done since the last write.

Precede all literal uses of %, #, and & with a backslash (for example, \%). Normally, **astgen** warns users of improper syntax, but the symbols listed above are not always checked by the program.

astgen(1)

Placing a character in the first column may cause cursor misplacement on the ASSIST forms.

When using **astgen**, the first column is reserved for internal use and should not be used by the user.

awk(1)

The known incompatibilities between the ARIX-OS 4.0 **awk** and ARIX-OS V.3 **nawk** (new **awk**) follow:

1. New keywords: `delete`, `do`, `func` (and `function`), and `return`. **nawk** programs may now contain user-defined functions.

2. New built-in functions: `atan2`, `cos`, `sin`, `rand`, `srand`, `gsub`, `sub`, `match`, `close`, `system`.

3. New predefined variables: `FNR`, `ARGC`, `ARGV`, `RSTART`, `RLENGTH`.

4. Parameters that appear before the first input file name are available to `BEGIN`. Until ARIX-OS V.3, however, they were not available until after the first input file was opened, which happened after `BEGIN` time.

5. The `for (i in arrayname)` construct may produce array elements in an order different from before.

6. The input field separator variable, `FS`, now specifies a regular expression, rather than just a single character. (The third argument to `split` also behaves this way.) Previously, **awk** only used the first character in the string assigned to the input field separator if the string had more than one character.

7. Strings may contain escape sequences like their C counterparts: `\b`, `\f`, `\n`, `\r`, `\t`, *\ddd*. Previously, the *\ddd* notation was not interpreted. Also, if a backslash was not part of an escape sequence, it was retained in the string. To be consistent with C, backslashes are dropped in **nawk** if they are not part of escape sequences. For example, **awk** prints **\c** for the command **print** "**\c**", and **nawk** prints **c**.

8. The precedence of operators has changed. **nawk** firmly establishes their precedence, which matches C precedence, but which differs from their previous **awk** precedence in some cases. Two examples of code that breaks are:

```
while ( n /= 10 > 1 ) ...

if (!"wk" ~ /bwk/) ...
```

The old version of **awk** is provided to avoid compatibility problems. See the section on "Future Directions" in these *Release Notes* for further information.

bc(1)

When the following command line is entered and **bc** cannot open *file2*, the error message displayed says that **bc** cannot open *file1*:

bc *file1 file2*

Also, when the following command line is entered and **bc** cannot open *file*, the error message displayed contains garbled characters instead of the name of **file**:

bc *file*

bc(1)

bc does not handle the following two constructs in the same way:

(1)
```
if ( expr ) {
...
}
```

(2)
```
if ( expr )
{
...
}
```

The first case produces what one would expect. The second case is equivalent to an **if** followed by an empty statement, and the compound statement always is executed. The second case dumps core silently.

)

The **bc** command prints an incorrect remainder when dividing with a nega- number.

```
$ bc
-1.2 % 1
9.8
<Ctrl-D>
```

1)

The utility **bfs** exits with normal values; in particular, a return value of 0 n a quit or an EOF is received.

cc(1)

The C compiler produces more efficient object code resulting in a faster runtime of programs recompiled with the new compiler.

cc(1)

To use the default **cc** to create a S90 executable from source, you need to hand off the **-Q** option to the linker–loader (that is, **ld**). To do this, use the **-W** option in the **cc** command line as follows:

```
/bin/cc -Wl,-Q <other options> tt.c
```

cdc(1)

The **cdc**(1) command ends abnormally when invoked without the **–m** option on an SCCS file that does not have the **v** flag set.

comm(1)

comm issues error messages when a comparison is made between two directories or if more than two files are compared.

cpio(1)

cpio can overwrite files when invoked with the **–u** option. An overwrite can also occur if the files in the archive are newer than the existing files.

If an I/O error occurs while reading the archive or if the file system runs out of space, the file being created is either corrupted or truncated, and the old existing file is lost because it has been overwritten.

cpio(1)

The utility **cpio** no longer causes bus errors where file_list is a list of more than 3000 files, all files start with ./, and none of the files are in the current directory.

```
cat file_list | cpio -ovB > /dev/rmt1
```

cpio(1)

The utility **cpio** now correctly handles the following situation:

```
dd if=/dev/rmt1 | tee file1 | cpio -idvBm
```

cpio(1)

Enhancements have been made to **cpio** to improve output performance to the 150 MB tape drive.

cpio(1)

The utility **cpio** no longer gives a usage message when the **–O** *file* option is used.

cpio(1)

The **–M** option of **cpio** performs as described in the documentation.

cpio(1)

On nine-track tapes, multiple requests cause the device to hang. When using a 9mt device, a prompt is returned immediately; however, the device is still rewinding. If another 9mt request is issued before the device finishes rewinding, the device will hang.

cpp(1)

Large programs were exceeding the maximum limit of characters in defines specified in **cpp**. This limit has been changed from approximately 36000 to 50000.

cpp(1)

The preprocessor, **cpp**, now gives a warning message in each of the following cases:

1. When calling macros, the number of formal parameters is different from the number of actual parameters:

```
#define a(x,y,z)        (x=y=z=1)
main()
{
        int c,d;
        a(c,d);
}
```

2. When no argument follows #ifdef or #ifndef:

```
main()
{
#ifdef
        exit(1);
#endif
        exit(1);
#ifndef
        exit(1);
#endif
        exit(1);
}
```

3. When #include is missing trailing separators > or " following the filename to be included:

```
#include <time.h
main()
{
        exit(1);
}
```

cpp(1)

The C language preprocessor, **cpp**, can compile complex macro definitions that previously gave the message "too much defining."

crontab(1)

The **crontab** files supplied in /usr/spool/cron/crontabs perform standard system performance and maintenance functions in areas such as uucp and disk usage. These files can be modified to be system specific.

crypt(1)

The **crypt** command now takes options that begin with a dash (–), such as **–k**. These can no longer be given as keys. This change only affects those who have the Security Administration Utilities (Domestic Overlay) package installed on their system.

csh(1)

The Berkeley 4.3 cshell, **csh**(1), has been added.

cu(1C)

When the **cu** command is used over a direct line, enter a carriage return to get a `login:` prompt. After the `login:` prompt appears, wait about 5 seconds before entering the login. If users do not wait, the first character entered is not displayed on the terminal screen even though it is accepted by the computer. This only happens when **uugetty** is used.

cu(1C)

The first invocation of **cu** after a power up may fail; the error message `cannot access device` is displayed. Later invocations of **cu** succeed.

1C)

If a user enters a **cu** from machine_A to machine_B, and then enters a **cu** ı machine_B to machine_C, the command ~**%take** *file* will not transfer *file* ıachine_A.

Transfer files over one link at a time by one of two methods.

- Login to machine_A; **cu** to machine_B and then to machine_C as described above. The command ~~**%take** *file* transfers *file* to machine_B. Then type

  ~~.

  From machine_B, use the command ~~**%take** *file* to transfer *file* to machine_A.

- If possible, **cu** from machine_A to machine_C. Files can be successfully transferred over a single link.

1C)

Occasionally, **cu** fails with the following message even though all devices available:

```
NO DEVICES AVAILABLE
```

cu(1C)

If there is a command in **/etc/inittab** that is respawning too rapidly and it generates an error message while the console user does a **cu** to another system, the console user will find that every command typed in is echoed twice on the console, although it is only executed once (as it should be).

date(1)

The **date** command no longer displays the bad format character message if the field descriptor is not recognizable. For example, the output of

```
date '+%e'
```

is the day of the month only, instead of

```
date: bad format character - e.
```

date(1)

The default timezone for **date** is now GMT. Another change affecting **date** involves the **TZ** environment variable. Previously, if **TZ** was unset or null, the timezone used would be **EST** or **EDT**. Now the timezone used is **GMT**. For example,

```
TZ='' date
```

now prints

```
Thu May  1 21:29:08 GMT 1986
```

instead of

```
Thu May  1 17:29:08 EDT 1986.
```

date also recognizes the extended syntax in the **TZ** environment variable for alternate time zones. This change should not cause problems since the old syntax continues to be recognized. The changes involving the **TZ** environment variable occur because the **date** command uses the new **cftime**(3C) function to compute the date and time. The **cpio**, **ls**, **mount**, **pr**, and **vi** commands also use **cftime**, and their functionality in displaying times is affected by the **TZ** environment variable in the same manner.

date(1)

date(1) has been changed to take into account the new date for Daylight Savings Time. .

dc(1)

The bus error that occurred if a **Y** was typed as initial input has been corrected.

ed(1)

The **ed** command now defaults to using the **/usr/tmp** directory to hold temporary files, instead of **/tmp**.

This change does not apply to the **vi** or **ex** programs.

No special action is necessary to use **ed** in single user mode if the **/usr** file system is not mounted. When **ed** cannot put its temporary file in **/usr/tmp**, it tries **/tmp** instead.

ex(1), vi(1)

The structure of the **/usr/preserve** directory used by **vi** and **ex** has changed.

Instead of saving an editing session as a file directly under the **/usr/preserve** directory, it is saved in a subdirectory with the name of the user whose session is saved. Only the same user can access the contents of the subdirectory.

In general, users will not be able to recover a **vi** or **ex** session preserved before an upgrade to ARIX-OS V.3. All sessions should have been recovered before upgrading.

ex(1), vi(1)

The **ex** and **vi** commands now exit with a return code equal to the number of errors encountered during the editing session. Before ARIX-OS V.3, no specified return code was used if errors were encountered.

Because all return codes must be between 0 and 255, if more than 255 errors are encountered the return code may not be accurate. If an integral multiple of 256 errors are found, then **ex** and **vi** exit with a zero return code.

ex(1), vi(1)

The **ex** and **vi** commands no longer set the eighth bit in the characters of the **%** expansion of the current filename.

When the percent sign, **%**, is used in a shell escape from **ex** or **vi** via the exclamation mark, **!**, the **%** is replaced with the name of the file being edited. Previously, each character in this replacement had the eighth bit set to 1 to quote it; now it is left alone.

Generally, older versions of the **ex** or **vi** commands on ARIX-OS V.3 can be used, but the percent sign cannot be used, **%**, in a shell escape via the exclamation mark, **!**, even if the file being edited has no special characters in it.

ed(1), vi(1)

vi no longer leaves defunct processes when using the ! command.

ed(1), vi(1)

vi no longer assumes that a file is modified because it contains a modeline.

file(1)

If an **a.out** was generated with the **-P** option of the **ld**(1), the **file** command reports:

```
Arete 68020 (paged)
```

Such an **a.out** will not execute on systems running any release prior to ARIX-OS V.3.

ftar(1)

A new utility, **ftar**, has been created to provide improved output performance to the 150 MB tape drive. **ftar** is identical to **tar** in all other respects.

getedt

The utility **getedt** is AT&T specific and is not supported.

help(1)

When using the help function and requesting information for "at", information for "ar" was given instead. This has been corrected.

help(1)

The help facility is a menu driven utility that provides assistance in areas such as defining terms, supplying examples for utilities, and getting started on a system. See the **help**(1), **glossary**(1), **helpadm**(1), **locate**(1), **starter**(1), and **usage**(1) manual pages for more information.

help(1)

The **help** facility's commands ignore SIGHUP and stay attached to a port that has been dropped or disconnected. This inattentiveness to SIGHUP makes the port useless; users cannot log in because there are two processes reading the port (**help**'s and **login**'s).

ipcs(1)

ipcs(1) always reports the number of processes attached to shared memory segments, NATTCH, as zero, even when running processes are currently attached to shared memory segments.

The information in the NATTCH column is incorrect. To obtain the correct information, follow these steps:

Step 1 Invoke **crash** as **root**.

Step 2 Type **od shminfo 3**. The last word on the line will be the number (in hexadecimal) of shared memory identifiers.

Step 3 Now type **od shmem 6**.

Step 4 Examine the first **shmem** identifier. If the leftmost digit of the third word is in the range 8 to f (hexadecimal) the identifier is in use, and users can get the address for the next step in the last word on the second line. If the leftmost digit is not 8-f, skip to Step 6.

Step 5 Type **od** <address> **3**. The left 4 digits of the last word is a hexadecimal number for the number of processes attached to this identifier.

Step 6 To go to the next identifier, add (hexadecimal) 30 to the address reported in response to **od shmem 6** and enter **od** <address> **6**.

Step 7 Repeat Steps 4-6 until all identifiers in use have been displayed. (The data reported by Step 6 will be all zeros.)

layers(1)

The AT&T layers/windowing capability is not supported.

ld(1)

If **ld -Q** is specified on the command line, system binaries are generated.

ld(1)

The utility **ld** generates both paged and nonpaged-format binary files without returning error messages.

ld(1)

A space is now accepted after the **-L** option.

ld(1)

The #hide and #export directives for **mkshlib** are now supported.

ld(1)

Files with 000 protection are no longer clobbered.

ld(1)

To create a S90 executable from an existing object module, use the **-Q** option. To make sure the correct libraries are linked in, use **-YL** and **-YU** in the command line.

/bin/ld -Q
<startup routine crt1.o>
<object modules>
<other options>
-YL,<first-default-path>
-YU,<second-default-path>
<routine crtn.o>

login(1)

The login ids for **rje** and **trouble** have been removed from the /etc/passwd file. These functions are not supported in ARIX-OS V.3.

login(1)

The utility login has been modified so that multiple uucp logins, all using /usr/spool/uucppublic for their home directory, will be able to change the .lastlogin file.

login(1)

ARIX-OS V.3 supports two logins, **login** and **login.any**. With **login**, root can only login on the system console; **login.any** allows root to login anywhere.

login(1)

A bus error problem has been corrected in **login** and **login.any**.

login(1)

After executing **exec login**, this warning message is produced intermittently: `no utmp entry ... execute from the lowest level shell` If the **who** command shows that someone is still logged in on that line, the **getty** must be killed which allows it to respawn. Otherwise, no action is required. The problem can be avoided by executing **exec su** – *xxxx* (where *xxxx* represents a login), or by logging out (hanging up or executing an **exit** command) and calling back.

login(1)

Environment variables are not lost when passed from **getty** to **login**.

lp(1)

If the **lp** command is issued on *file* within a directory that has 700 permissions, the following error messages are displayed:

```
lp: can't access file file
lp: request not accepted
```

cat or **pr** *file* and pipe the output to **lp**.

ls(1)

The utility **/bin/ls** reports incorrect information when using the **-s** option and the file is over 138K in size.

ls(1)

Now reports in 512-byte blocks.

mail(1)

The mail for root will no longer fill up with cron messages like the following when the uucico link has not been used. Therefore, the .Log directory is empty.

```
grep: can't open /usr/spool/uucp/.Log/uucico/*
```

mail(1)

In order for **mail** to use forwarding correctly, there must be an entry for mail in /etc/group.

mail(1)

A problem causing empty files to be removed when running the **mail** program has been corrected.

mailx(1)

mailx changes the modification time of the mailbox (**/usr/mail/***login* so that the timestamp on the mailbox reflects the last time new mail arrived, not the last time the mailbox was accessed and changed.

mailx(1)

Using the command **mailx –f** *a/b*, where *a* is a nonexistent directory, causes **mailx** to pause for a long time, as it continuously tries to access the nonexistent directory. Eventually, **mailx** will time out.

The argument supplied to the **mailx –f** option must be a valid directory.

mailx(1)

The **mailx** command now utilizes the tunable parameter NOFILES to represent the maximum number of open files per process. This allows **mailx** to open as many files as are configured on the system. This version of **mailx** does not run on any release prior to ARIX-OS V.3.

make(1)

Now **make**(1) returns the correct part of the variable with its suffix modified when the modifiers "D" and "F" are used in combination.

mkdir(1)

The *mkdir* utility no longer returns a non-zero value on the completion of successful system calls.

mvdir(1)

The utility **mkdir** now accepts commands of the form `mvdir x xy` (for example, `mvdir tmp tmp/junk`). Previously the error message "arguments have common path" was given.

nawk(1)

&& and || have been fixed.

nawk(1)

The utility **nawk** is a new version of **awk**. This new utility contains the following enhancements:

- support of 8-bit code sets
- the ability to define functions
- new keywords: delete, do, func, function, return
- new built-in functions: atan2, cos, sin, rand, srand, gsub, sub, match, close, and system
- new predefined variables: FNR, ARGC, ARGV, RSTART, and RLENGTH
- input field-separator variable, FS, and the third argument to split are treated as regular expressions
- precedence of operations now matches C language precedence

Because some of these enhancements may not be compatible with existing **awk** programs, the user will get the old version by typing **awk**.

nvram(1)

The utility **nvram** is an AT&T specific utility and is not supported.

od(1)

If a file has an odd number of bytes, **od –c** reports a trailing null byte.

pg(1)

The utility **pg** no longer hangs when searching for the end of a very large file. If the file is too large, an error message is returned.

pr(1)

When **pr** is used to print a file containing lines greater than the column width, 80 (for example, **pr –n –i –w80 –l60 0f** *file* | **lp**), the printer prints a new line and carriage return <CR> and then prints the rest of the line. This causes the printer to print more than the specified number of lines on a page. If more than four or five lines wrap around in this manner, the form feed issued after printing the page causes a blank page to appear in the listing.

pr(1)

The **pr** command now correctly interprets the combined options **–m –k** as an error, where before one option would be ignored. Shell scripts that took advantage of the earlier fault in the **pr** command must be changed to use the correct option.

pr(1)

The command:

```
pr -e file >file.out
```

correctly expands the first tab seen on a page. It no longer adds an extra space.

prof(1)

The utility **prof** produces an error message when the options **–a** and **–n** are used together or if **–o** and **–x** are used together.

ps(1)

The **ps** command now correctly interprets a non-numeric argument to the **–g** or **–p** options as an error, where previously it was treated as zero.

Change any shell scripts that rely on the previous incorrect behavior of the **ps** command to reflect the correct operation.

Note that the "**–g**" option is used to examine the processes belonging to a particular process group leader, not to a particular user group identifier.

ps(1)

The **ps** command no longer prints out a fake wait condition.

ps(1)

The **ps -g 0** and **ps -p 0** commands no longer return an error condition. They now return information on process 0 instead.

ps(1)

The process state indicated is now correct.

sar(1)

The utility **sar** now reports correct totals when using the **–r** option.

sar(1)

The **sar** and **sadc** utilities have been modified to correctly report information on a per CPU basis. As well, the disk activity is accurately reported.

sar(1)

sar produces a table in which CPU time is represented as user, system, waiting for I/O, or idle. These figures may not total to 100%. The unaccounted time is spent waiting for a runnable process to be loaded into memory.

sar(1)

Activity greater than 100% now rounds to 100%.

sar(1)

The output format is now more correct.

sdiff(1)

When doing an **sdiff**, if one of the files contains more than 196 characters in one line, the **sdiff** output loses the separator symbol (|) or loops infinitely, or both.

Do not **sdiff** a file that contains more than 196 characters in a single line.

SGS

Floats are not passed correctly to functions in C programs in some cases. The following program is an example of the problem:

```
main(){
     float f;
     f = 12.5;
     printf("main: f = %f\n", f);
     one(f);
     two(&f);
}
one(f2)
float f2;
{
     printf("one: f2 = %f\n", f2);
     two(&f2);
}
two(f3)
float *f3;
{
     printf("two: f3 = %f\n", *f3);
}
```

This programs outputs:

```
main: f = 12.5
one: f2 = 12.5
two: f3 = 2.640625
two: f3 = 12.5
```

The output call of two() from one() is wrong, it should print 12.5. The following occurs:

```
main() sets f = 12.5
main() then calls one() with the VALUE of f equal to 12.5.
```

However, a function is always called with the next larger type. In other words, one() is really called with a double, not a float. one() realizes that it has a double, and does the appropriate things with it. one() then calls two() with the address of f2 which is the address of a DOUBLE, not a float. Since just the address is passed, two() doesn't know that it got the address of a double instead of a float. At this point, it tries to print the first word of a double as a float, and the output is incorrect.

The workaround is to use doubles instead of floats -- either change all floats to doubles, or use #define float double at the beginning of the programs, and recompile.

SGS

The loader (ld) can't read an archive file after the last member has been removed. Below is a sequence of shell commands to reproduce the problem:

```
$ echo 'main(){}'>mojo.c; cc -c mojo.c
$ ar cvq libmojo.a mojo.o
q - mojo.o
$ cc libmojo.a              # it works here
$ ar dv libmojo.a mojo.o
d - mojo.o
$ cc libmojo.a
ld fatal: can't read archive header from archive libmojo.a
$ ls -l libmojo.a
-rw-rw-r--       1 whp   nap              8 Oct 3 16:11 libmojo
$ od -bd libmojo.a
0000000        08508  24946  25448  15882
           041 074 141 162 143 150 076 012
0000010
```

One possible workaround is to put a dummy ".o" file into the archive library. Another is to keep updating makefiles.

SGS

The C compiler does not give an error message for an undefined structure until a member of that structure is referenced. The following is an example:

```
main()
{
    extern struct adr record;
    record.hello = 0;
}
```

The compiler gives the following error messages:

```
"ccompiler.c", line 4: hello undefined
"ccompiler.c", line 4: structure/union member required
"ccompiler.c", line 4: illegal lhs of assignment operator
```

Lint does give a warning.

SGS

The compiler now properly handles the logical and/or cases in which no **if** or **while** statement is involved:

```
main()
{
int a,b;
a=b=1;
((a=(a==2)) && (b=3));
printf("%d %d0,a,b);
}
```

SGS

The compiler now properly handles null dimensioned array:

```
main()
{
int a[][];
exit(1);
}
```

SGS

The compiler now properly simplifies constant expressions involving the operators &&, ||, and the relational operators.

The c compiler fails with "illegal initialization" on similar constructs to "static int x = 1 == 2.5;" and all of the following comparisons:

```
(int, long int == != > < >= <= && || double)
(double == != > < >= <= && || int, long int)
(double == != > < >= <= && || double )
(evaluation order || / &&)
(|| && group left to right)
```

Example 1:

```
main()
{
    static int var = 6 && 4;
    if (var != 1)
        exit(1);
    exit(0);
}
```

Example 2:

```
main()
{
    static int var = 5 || 0 && 0;
    if (var != 1)
        exit(1);
    exit(0);
}
main()
{
    static int var = 1 && 0 | 3;
    if (var != 1)
        exit(1);
    exit(0);
}
```

Example 3:

```
main()
{
    static int var = 0 && 5 / 0;
    if (var != 1)
        exit(1);
    exit(0);
}
```

SGS

The compiler now allows a typedef name to be redeclared in an inner scope. For example:

```
typedef int i;
main()
{
    int i;
    i = 2;
    if (i != 2)
        exit(1);
    exit(0);
}
```

SGS

Previously, the method by which the compiler generated labels could result in multiply defined labels. Now the compiler appends the nameid with a '%' followed by the function number to create a unique label ('%' is an illegal C label).

SGS

The optimizer now correctly supports the special registers in connection with the Floating Point Processor.

SGS

Now the optimizer no longer complains about the special register %iaddr.

SGS

For the 68020 C compiler, code using double indirections to type float no longer result in incorrect code. An example follows:

```
struct info {float *fptr;}

    main()
{
    struct {float a;} c;
    struct info sptr, *structptr;
    structptr=&sptr;
    c.a=12345.67;
    sptr.fptr = &c.a;
    printf("*structptr->fptr = %f0,*structptr->fptr);
}
```

SGS

According to the the C rule "If the type-specifier is missing from a declaration, it is taken to be int." The following program now compiles:

```
#include <stdio.h>
char x[10],y[10];z[10];   /*two ";", should fail*/
main()
{
printf("%s, %s, %s0,x,y,z);
}
```

Page 193 in section 8.2 of the book *The C Programming Language* should be read for a complete description.

SGS

Programs whose text is exactly a multiple of a pagesize no longer cause a bus error on execution. On linking, their textsize is rounded up by four bytes allowing room in text for instruction lookahead on ARIX computers.

SGS

The compiler was not removing the "/tmp/inl*" (inline code) files when it exited normally or due to error. Now, extra "inl*" files no longer are left in the /usr/tmp and /tmp directories.

SGS

The "cc -g" command no longer causes the debuggers to look for the wrong file.

SGS

Conversions of 2 integers $>= 2^{31}$ from a floating point value into an unsigned long integer can only be done by hand due to the lack of machine instructions.

sh(1)

Using **sh** || does not work correctly with built-in commands. For example, the command sequence

cd *nowhere* || **date**

generates an error and does not execute **date**. This example can be executed without using ||.

```
cd nowhere
if [ "$?" != "0" ]
then
date
fi
```

sh(1)

When the command **sh –c** *string* is executed, the flag value returned is blank. The correct value is **flag=s**. If the **–c** option and the *string* are piped into the shell command, the correct value is returned.

Do not try to execute the **sh** command using the **–t** and **–c** options together. It does not work.

sh(1)

A trailing colon in the shell PATH variable causes the current directory to be included in command searches. Previously, a trailing colon was ignored.

sh(1)

The **test** command now uses the effective user and group IDs to determine permissible file access, instead of the real IDs.

The only way to invoke the **test** command with different effective and real IDs is to invoke the command, or a shell script containing it, from a compiled program that has the set user (group) ID on execution permission. Otherwise, the effective IDs are the same as the real IDs, and this change has no effect. If the program relies on the test operators to behave as they did previously, which is to have **test** use the real user and group IDs, it should be changed to use the **setuid**(2) or **setgid**(2) system calls to set the effective ID to the real ID before invoking the **test** command or shell script.

sh(1)

The shell no longer treats the eighth bit in the characters of a command line argument specially; it also no longer strips the eighth bit from the characters of an error message.

If there are any programs that set the eighth bit of characters, they should be changed. Use one of the standard shell quoting mechanisms, such as the backslash, instead of setting the eighth bit.

sh(1)

The shell **type** command displays backslashes before each character that was quoted initially.

This change is a result of the change described above, where the eighth bit is no longer used by the shell to quote characters.

sh(1)

The result of a parameter substitution in a command like

ls "*${a:=xyz abc} lmnop*"

is now correct.

In general, the parameter substitution

${ parameter:=word }

when used inside double quotes and when the *word* contains spaces, now works correctly. If there are programs that rely on the previous incorrect behavior, they should be changed to reflect the correct behavior.

sh(1)

The **sh** variable SHACCT can now be used to store the local shell accounting record.

shl(1)

If a user hangs up while in shell layers, **/etc/utmp** may not be updated (that is, the **who** command still shows that the user is logged in, and the **ps** command shows a **getty** running on that line). If someone else then calls into that line, the **login** fails because the **utmp** entry cannot be found. To kill the **getty** corresponding to the affected line, follow this procedure:

Step 1 Execute **ps -ef** to find the process ID of the **getty** running on the line erroneously reported as occupied.

Step 2 Execute **kill -9** *PID* to kill the **getty**. When the **getty** automatically respawns, the problem is cleared.

shl(1)

When using the **shl** command, trapping a "hang up" signal inside a layered shell suspends the layer. This renders the device unusable until the shell is killed. A warning message stating that a layer is still running is displayed, and then the user is returned to the ARIX-OS V.3 system shell.

If the terminal does not accept input commands because of the suspended layer, log in at another terminal. To kill the suspended shell, follow this procedure:

Step 1 Find the process number of the shell by executing the following command:

ps -eaf

Step 2 Look at the output of the command for the appropriate pseudo tty (for example, sxt001) and for the process number of the shell.

Step 3 Enter the following command:

kill -9 *PID*

where *PID* is the process number of the shell to be killed.

shl(1)

If a user is at the console in **shl** and the following sequence of commands is executed, the shell layer does not respond properly:

1. an **stty** command with a carriage return select style argument (for example, **stty cr3**)
2. an **echo** command
3. an **stty** command with a carriage return select style argument (for example, **stty cr0**)

In particular, the layer's prompt does not return until the <BREAK> or <DEL> key is hit. Once the prompt appears, the output of any command executed does not print fully or at all until one or more carriage returns are entered.

Return to the **shl** control layer and use the **shl** delete command to delete the layer.

shl(1)

If a background process is executed not in a shell layer that sends output to the terminal, and then shell layers (**shl**(1)) are entered, some screen output from the original process may be lost. When **shl** is entered, the output from the original process temporarily stops printing on the screen. The screen output resumes when **shl**(1) is exited. Loss of data, if it occurs, is noticeable when the screen output resumes.

This is not normal use of the system, and should be avoided.

shl(1)

When resuming a layer of **shl**, the `resuming` *xyz* prompt is often garbled.

sort(1)

The sort command used with the "merge" option aborts upon inputting a line longer than 512 characters if the -zrecsz option is not specified. This is because the system default maximum line length is 512. The manual page (**sort**(1)) states that an option -zrecsz is available for supplying the size of the longest line.

> "The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the -c or -m options, a popular system default size will be used. Lines longer than the buffer size will cause sort to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination."

stty(1)

The command **stty sane** will now restore the terminal settings to a usable state.

stty(1)

The **stty -raw** command no longer changes character size from CS8 to CS7 nor changes settings for PARENB. It now enables input processing for the following special characters: ERASE, KILL, INTR, QUIT, SWTCH, EOT, and enables output post processing.

tar(1)

When using the utility tar, the 'xf' options would not work without the "b" option. Now the block size is determined automatically when reading tapes on raw magnetic tape with "f" option and the "b" option is not required.

uucico(1C)

On occasion, uucico generated divide-by-zero errors. This has been fixed.

uucp(1C)

In the **/usr/lib/uucp/Permissions** file, read/write/execute permissions are defined for each **uucp** login. These permissions authorize remote hosts to read/write/execute local files using **uucp** and **uux** commands. Currently, a file is not readable by giving it read permission in the **Permissions** file. To make a file readable, both read and write permissions should be given.

uucp(1C)

uucp and **uux** are supposed to allow the use of a tilde (~) with a login name to designate a user's home directory (for example, ~phyllis for phyllis' home directory). This works in most cases, but **uux** does not handle tilde expansion in **/usr/lib/uucp/Permissions** file correctly.

The full pathname should be used for the home directory in the **Permissions** file.

uucp(1C)

uucp logins, such as **uucp** and **nuucp**, should have distinct user ID numbers. In the **/usr/lib/uucp/Permissions** file, different read/write/execute permissions are defined for each **uucp** login. Login names in this file are used to associate permissions with the login. These permissions do not work correctly if more than one **uucp** login has the same user ID. For example, if **uucpa** and **uucpb** both have user ID *10*, and **uucpa** appears before **uucpb** in the **/usr/lib/uucp/Permissions** file, then whether **uucp** logs in as **uucpa** or **uucpb**, the permissions will be those of the first user ID (**uucpa**) that matches the user ID requested. Logging in as **uucpb**, which has user ID *10*, means that the permissions will be those of **uucpa**.

To avoid this problem, use distinct user ids for **uucp** logins.

uucp(1C)

The **uucp** command now utilizes the tunable parameter NOFILES to represent the maximum number of open files per process. This allows **uucp** to open as many files as are configured on the system. This version of **uucp** does not run on any release prior to ARIX-OS V.3.

uucp(1C)

In Chapter 9 of the *System Administrator's Guide*, "Basic Networking", the section "Supporting Data Base" gives an example of the *time* field of the **Systems** file used by **uucp**:

```
Wk 1700-0800, Sa, Su
```

For **uucp** to work properly, the spaces should be removed. The corrected example reads:

```
Wk1700-0800,Sa,Su
```

Do not separate subfields with spaces in the **time** field of the **Systems** file.

uucp(1C)

uucp preserves execute permissions across transmissions, and gives 0666 read and write permissions. Files copied by uucp are owned by uucp.

If a user wants to use **lp** to print a file on a remote machine after **uucp**ing the file to the remote machine, the use id for **lp** must have read permissions in all the directories in the path leading to the file to be printed. /usr/spool/uucppublic provides these permissions.

uucp(1C)

In order for the **cu** command to work more reliably, two dashes should be used after the phone number to force a longer delay in the calling sequence.

uucp(1C)

When trying to set the retry time to less than 5 minutes, the retry time was automatically increased to 5 minutes. Now the retry time can be set to less than 5 minutes.

uucp(1C)

The utility **login** has been modified so that multiple **uucp** logins, all using /usr/spool/uucppublic for their home directory, are able to change the .lastlogin file.

uucp(1C)

The group ids for the logins "uucp" and "nuucp" have been changed to match those numbers set up in the group file.

uugetty(1C)

The utility **uugetty** now supports timers in the same manner as **getty.**

uulog(1C)

If the **f** or **number** option is specified with **uulog** (see **uucp**(1C)), a prompt is not given after the execution of **uulog**. To regain access to the terminal, press the <BREAK> key.

uuto(1C)

The utility **uuto** no longer makes extra directories when sending directories through **uucp**.

what(1)

The **what** command, used to identify SCCS files, currently recognizes the identification string "@(#)". Other miscellaneous strings are not recognized.

who(1)

The utility **who -q** was not returning the correct value on successful execution. Zero is now returned.

who(1)

The **who –q** command now lists the login names in space padded fields of equal size and no longer sorts the entries.

If there are any programs that process the output of the **who –q** command, they should be inspected to see if they will still work with the new form of the output.

who(1)

The command line **who am i** returns the first eight characters of the user login ID. For example, if the user's login ID is "user123456," the string "user1234" is returned.

System Administrator Commands

acctsh(1M)

The /usr/lib/acct/ptecms.awk and /usr/lib/acct/ptelus.awk files are no longer executable.

bad block

The Bad Block Handling utilities, **hdeadd**, **hdefix**, and **hdelogger**, are not supported in this release.

chrtbl(1M)

A file called ASCII is installed in /lib/chrclass to provide an example input file for users who wish to use the new character classification utility, **chrtbl**.

ckauto(1M)

The utility **ckauto** requires the support of loadable drivers for operation. Loadable drivers are not currently supported, therefore **ckauto** is not supported in this release.

config(1M)

The **config** utility has been modifed to support the buffer type expected by the device driver. Use the value 0400 in the /etc/master file driver mask to enable the new buffer. Absence of mask 0400 indicates the driver requires compatibility buffer.

crash(1M)

While using the -u option of **crash**, the problem that caused garbage characters to print after the statement "file modes():" and occasionally caused the terminal to lock up has been removed.

crash(1M)

When using the "proc" option of crash, the name of the event is now properly printed.

crash(1M)

The **crash** command now utilizes the tunable parameter NOFILES to represent the maximum number of open files per process. This allows **crash** to open as many files as are configured on the system. This version of **crash** does not run on any release prior to ARIX-OS V.3.

crash(1M)

When the crash command is executed, if the inode function is specified with a value around 300 or more, either the correct information or the "Inode out of range" error message now displays instead of the system hanging.

crash(1M)

When the command "f 32" followed by "i <inode of the 'f 32' command>" is entered, the string "%s%s%s%s%s" was displayed. This problem has been corrected.
The **crash** command has been modified for STREAMS.

cron(1M)

The following message means that during shutdown, **cron** processes are killed:

```
cron aborted: SIGTERM
```

This message is normal.

cron(1M)

The intermittent problem of **cron** not executing all files in /usr/spool/cron/crontabs has been resolved.

cron(1M)

The **cron** startup script does not automatically truncate the **/usr/lib/cron/log** file. Therefore, when this log file reaches the ulimit value, **cron** logging becomes inoperative. In addition, the log file also takes up a large amount of disk space in **/usr**.

The **/usr/lib/cron/log** file can be periodically truncated by modifying the **cron** startup script.

ctrace(1M)

ctrace can now properly display negative integer values. Previously, the "`Bus error -- core dumped`" message was displayed.

dcopy(1M)

The usage message issued by **dcopy** does not list all options appropriately.

When summarizing, **dcopy** prints the new gap and cylinder sizes when it says that it is printing the old information. The new sizes, however, are printed correctly.

dcopy(1M)

dcopy –f recreates the file system based on the `fsize:isize` count given; however, the blocks specified are assumed to be logical blocks (physical blocks = 2 x the number specified), and the inode count is assumed to be 16 x the number of inodes requested. Thus, **dcopy** does not accept the inode or block count specified to be the actual number of inodes or physical blocks required, as other commands do (for example, **mkfs**).

dcopy(1M)

The utility /etc/dcopy now correctly handles processing the superblock information.

dd(1M)

If **dd** is invoked with the parameter cbs (conversion buffer size) greater than zero, the command no longer fails and the error message "`dd: write error: Not a typewriter`" does not appear.

dd(1M)

When writing to tape using **dd**, the correct amount of data is now being transferred. The problem that prevented the buffers from being dumped correctly has been fixed.

dd(1M)

The *seek* function of **dd** no longer truncates data in the output file.

/dev/console

When a system is delivered, /dev/console and /dev/syscon are two separate devices. When the system is brought into multi-user, then shut down, /dev/syscon is linked to whatever device the system is being shut down from (usually /dev/console). From that point on, /dev/syscon and /dev/console remain linked until either one of the devices is removed or the system is shut down from a terminal other than the console.

devinfo(1M)

The **devinfo** utility, run only by superuser, is used to print device specific information about disk devices. Using the options specified in the man page, the user can obtain information that pertains to a particular disk, or a specific partition on the disk.

Some of the information that is printed in the 3B2 version of the **devinfo** utility is not applicable to this system, i.e. Software version, Drive Identification Number, and Partition flag.

devinfo(1M)

devinfo no longer prints a usage message in response to normal usage.

devinfo(1M)

Now recognizes SCSI devices.

df(1M)

When using the utility **df**, the **–f** option now works correctly.

df(1M)

The utility **df** correctly interprets the specified directory when "." and ".." are used. The full pathname of the specified directory no longer needs to be used.

df(1M)

In heterogeneous networks with an ARIX system functioning as the server, the **df** command failed intermittently. This has been corrected.

df(1M)

Now reports in 512-byte blocks.

disk setup files

The files found in the directory /etc/vtoc are description files that contain the default setup values for the various supported disks. The file name convention is as follows:

hd###dft

where *hd* stands for hard disk, ### is replaced by the size of the disk (i.e. 337 for the 337 MB drives), and dft stands for default.

disks(1M)

The **disks** utility, run only by superuser, checks to see what devices are hooked to the controller boards. When a device is found, **disks** checks the /dev directory to verify that the appropriate device file is present. If a disk device file is missing, it is created; if the device file is incorrect, it is recreated. Device files are recreated if the file should be either block/character; device files are not recreated if the minor number or owner, group, or mode is incorrect. The utility reports any errors as well as any new devices that are created. Cartridge and nine-track tape devices are no longer made with **disks**. The utilities **sysadm mkrmt** and /local/bin/mkrmt should be used to create cartridge tape devices. **sysadm mk9mt** or /local/bin/mk9mt should be used to create nine-track tape devices.

disks(1M)

Block devices are now created correctly by **disks**. Previously, block devices were created as character devices.

disks(1M)

/etc/disks now makes make devices for drive 3 when drives 0, 1, and 3 are connected to controller 0 although drive 2 is missing.

diskusg(1M)

diskusg no longer interprets the comment character '#' as a special file to be processed.

drvinstal(1M)

The utility **drvinstal** requires the support of loadable drivers for operation. Loadable drivers are not supported therefore "drvinstal" is not supported in this release.

du(1M)

When **du** is invoked from the root directory, the extra '/' is no longer prepended to the path names printed.

errdemon(1M)

The errdemon/error reporting functions have been added to the release. The errdemon is started when entering multi-user state. Any errors are logged in the /usr/adm/errfile file. The **errpt** utility will read this file and print a formatted error report.

expreserve(1M)

When going into multiuser state, the files in /tmp were being cleaned up before the utility /usr/lib/expreserve was being executed. The order is now set up properly.

find_reqs(1M)

An ARIX-OS V.3 utility, /local/bin/find_reqs, has been added to the release (see the manual page for **find_reqs**(1M)).

fltboot(1M)

The utility "fltboot" is an AT&T specific utility and is not supported.

fmtflop(1M)

The utility fmtflop deals with formatting floppy diskettes. This utility is not supported because ARIX does not support diskettes.

fmthard(1M)

The utility fmthard is not supported in this release.

fsck(1M)

If **lseek** is used to skip over entire blocks of a file being written, the kernel assigns a block number of 0 to the skipped blocks. Reads of any bytes in those blocks correctly return a value of 0. However, because the number of "non-zeroed" blocks allocated to the file is consistent with the length of the file, an **fsck** of the file system produces the following message:

```
POSSIBLE FILE SIZE ERROR
```

Core dumps sometimes occur in these circumstances.

There is no damage to the file system. If the file size error messages are bothersome, they can be eliminated by determining the name of the file corresponding to the inode that has the "possible error" with **ncheck**(1M) and, after mounting the file system, copying that file to a temporary file and then back to its original name. Blocks are allocated on the copy to hold all the bytes with values of 0.

fsck(1M)

If **/tmp** is filled up with enough file names to run the **root** file system out of inodes, an **fsck** of the file system can produce a `POSSIBLE DIR SIZE ERROR` message. The file system itself is not damaged.

: the error messages are bothersome, they can be eliminated by
ng **init 1**, moving the contents of the directory that
)kes the message to a temporary place, removing and recreating
irectory, and then moving the files back:

```
cd /
mv tmp badtmp
mkdir tmp
find badtmp -print | cpio -pdv tmp
rm -rf badtmp
init 2
```

ıot change directory modes and permissions or the modes and permis-
ı of the files in that directory.

ı(1M)

There is no longer a bus error when the **0**, **x**, or **e** format options are speci-
The first data item at the specified address is "0" and only one item is
ed.

ı(1M)

Decimal words are printed with the **-e** option of **fsdb** instead of the hexa-
nal words previously printed.

ır(1M)

Root privilege is now required to kill a process that was not executed by
ıser.

ır(1M)

The utility **/etc/fuser** now works properly on both 12.5 MHz and 25 MHz
boards.

fuser(1M)

fuser –k *file1 file2* does not kill all processes using *file2*. It will kill the processes using *file1* (as specified on the manual page), but the **–k** flag is turned off before the processes using *file2* are found.

fuser –k *file1* **–k** *file2* works correctly.

fuser(1M)

The **fuser** command now utilizes the tunable parameter NOFILES to represent the maximum number of open files per process. This allows **fuser** to open as many files as are configured on the system. This version of **fuser** does not run on any release prior to ARIX-OS V.3.

getmajor(1M)

The **getmajor**(1M) utility will print the slot numbers along with information such as the type, board, and status (see **getmajor**(1M) manual page). The usage of getmajor is as follows:

getmajor -a | name

where *-a* will print all of the devices in the configuration and name will print only that device if found. Devices are in the form of /dev/icbxx.

getty(1M)

The utility /etc/getty now supports the gc timeout labels.

getty(1M)

If a **getty** is spawned from **/etc/inittab** on a non-existent terminal, it will fail after printing an appropriate error message. However, if this **getty** fails before the console **getty** opens the console terminal, it will be the first process to open it, and the console becomes the controlling tty for the failed **getty**. This action prevents the console **getty** from getting the console as the controlling tty, and users are prevented from logging into the console with the following error message `Login incorrect`.

To correct this problem, log in to the console or one of the ports, edit **/etc/inittab** to turn off the **getty** producing the error message, and enter:

telinit q

holidays

The holidays file in the accounting package now accurately reflects the holiday schedule for 1988.

icb(1M)

icb-l now reports cpu speed.

icb(1M)

8/16/32 MB MEM boards are now recognized when they're populated with only 8 MB.

init(1M)

init S drops the remote line and changes speed to 9600 baud when entered from a remote terminal.

Only use **init** from the console.

init(1M)

Executing **init s** from within **shl** layers causes inconsistent results. Sometimes, the machine may hang after printing the following message:

```
init: single user mode
```

Other times, the system may not really change run states. At no time within **shl** layers does **init s** do what it is supposed to do.

init(1M)

When users go to **init 3** state, the listener is already active; thus, the following message appears on the console:

```
/usr/bin/nlsadmin: listener already active
```

Ignore this message.

On the other hand, when the system is shut down, the listener is killed in more than one spot; thus, a message goes to the console that the listener is not running the second time.

init(1M)

The **init** command now reads an environment file (**/etc/TIMEZONE**) when it first begins to run. It uses the contents of that file to retrieve default environment variable settings, and it passes these settings into the environment of all processes it spawns.

The **init** command now reads the file **/etc/TIMEZONE** to retrieve the **TZ** environment variable setting for the system it is running on. The retrieved **TZ** value as well as a default **PATH** value are then passed into the environment of every process spawned by **init**. Formerly, the only environment information passed to processes spawned by **init** was a value for **PATH**; all other environment variables were left uninitialized. Now, up to five environment variables can be specified in the **/etc/TIMEZONE** file. This provides the ability to specify system-wide default values for environment variables, including values that may be useful for international systems, such as default character sets.

init(1M)

The **init** command now utilizes the tunable parameter NOFILES to represent the maximum number of open files per process. This allows **init** to open as many files as are configured on the system. This version of **init** does not run on any release prior to ARIX-OS V.3.

init(1M)

Powerfail handling has been enhanced in ARIX-OS V.3. Now, when init receives a power fail signal (SIGPWR) from the kernel, it confirms that the shutdown is imminent. If it is not, the signal is ignored, and a message is sent to the console. Otherwise, init will check if any powerfail actions (powerfail or powerwait) were found in the boot-time read of /etc/inittab. If so, they are executed. If not, init will try to execute /etc/powerfail. If init has nothing it can do, it tells the kernel to do its own handling of the powerfail condition, which kills running processes and syncs the disks before powering off the system. The init utility then gives control back to the kernel.

The utility /etc/init also recognizes powerfail when in single user mode. In addition, an entry must be made in /etc/inittab like the following line:

```
pf:s:powerfail:/etc/powerfail 1> /dev/console 2>&1
```

The "*s*" in the run level field represents single user mode. If no run level is specified, then the entry is valid for run levels 0 through 3. Therefore, the following entries allow the user to specify a powerfail script for run levels 0-3 and another script for single-user state.

```
pf::powerfail:/etc/powerfail 1> /dev/console 2>&1
pf:s:powerfail:/etc/singlepower 1> /dev/console 2>&1
```

There MUST be two powerfail entries, like above, in order for powerfail to work in both single user mode and in run levels 0-3.

The /etc/shutdown script has been modified to support powerfail. As well a default powerfail script has been added that upon powerfail will shutdown the system. In order to turn off the power supply after a power failure, the user must issue a "kill -19 1" command. This command can be put in the powerfail script as the last statement.

The powerfail scripts, powerfail.sh and singlepower.sh, have been modified to issue a "kill -19 1" at the end.

init_disks(1M)

The script **init_disks** has been added to Release 2.2 to ensure the disk device nodes have been make correctly.

killall(1M)

Previously, /etc/killall would kill the getty and then init would respawn the process. Now "init s" is executed prior to /etc/killall in /etc/shutdown.

labelit(1M)

If an improper device is specified, the following error message is displayed:

```
not a character-special...
```

led(1M)

The utility led is an AT&T specific utility and is not supported.

lost+found

A script, /local/bin/lost+found, has been added to the release. This script will help set up the lost+found directory that should be present on each file system. The script should be run after making a new file system.

mk9mt(1M), mkdsk(1M), mkrmt(1M)

The utilities to create device nodes for disk and tape, **mk9mt**, **mkdsk** and **mkrmt**, are now installed in the /local/bin directory for direct access as well as being available through the **sysadm**(1) package.

mk9mt(1M), mkdsk(1M), mkrmt(1M)

The utilities **mk9mt**, **mkdsk** and **mkrmt** set up mode, owner and group correctly.

mkboot(1M)

The utility mkboot is not supported in this release due to the absence of support for loadable drivers. The utility has no replacement.

mkfs(1M)

The mkfs command will now warn the user if the file system being operated on is mounted. The default STEPSIZE for disks larger than 168 MB has been changed to 7.

mkfs(1M)

The mkfs utility now supports SCSI disk drives.

mkfs(1M)

Running **mkfs** on a mounted file system would remake the file system. This has been correctd to work for unmounted file systems only.

mktty

Permissions are now

owner=root
group=sys

mkunix(1M)

The utility mkunix is an AT&T specific utility and is not supported.

mount(1M)

When a **mount** is done, the permissions used for directory access to the mounted file system are those of the mount point directory. The visible permissions, when executing **ls –al**, at the mount point are the permissions of the ".." directory in the mounted file system, and are misleading. For example:

- the mount point directory before mounting the file system:

  drwxr-x--- mnt(permissions 0750)

- the entry in the file system to be mounted:

  drwxrwxrwx (permissions 0777)

- after the file system is mounted on the mount point directory, the visible permissions are:

  drwxrwxrwx mnt(permissions 0777)

  while the real permissions are:

  drwxr-x--- mnt(permissions 0750)

Thus, any program fails that attempts to read **/mnt/..** after the file system is mounted, since read access is denied because it is based on the real permissions of the mount point directory.

mountall(1M)

The **mountall** command now handles large file systems.

newboot(1M)

Specifications of controllers and disks are now handled properly.

newboot(1M)

Some error handling of **malloc** return values have been corrected.

newboot(1M)

The newboot utility, run only by superuser, loads a new bootimage onto the designated reserve area. The usage is as follows:

```
/etc/newboot [-y] from_location to_location
```

where the from_location is the tape drive or file name that describes the new bootimage location, and to_location is the reserve area where the bootimage is copied. The -y option tells the program that the user is sure about loading this new bootimage. Otherwise, newboot asks for confirmation before copying the new bootimage to the reserve area. For example:

```
/etc/newboot /dev/rmt1 /dev/rv0
```

NOFILES

In releases prior to ARIX-OS V.3, the system tunable parameter NOFILES (total number of open files per process) had a maximum value of 64. In Release 2.2, NOFILES can be set to any number that benefits system performance. Refer to the "Performance Management" section of the *System Administrator Guide* for a more detailed explanation.

NOFILES

When the NOFILES parameter is set to 64, a file table overflow occurs if three processes that open the maximum number of open files are executed. When incrementing the NOFILES parameter to 64, the NFILE parameter should also be increased by 2.5 to 3 times the NOFILES parameter. This will alleviate any file table overflows since NFILE's default value is 150.

passwd(1M)

The **sys** login in **/etc/passwd** has its current directory in **/usr/src**. Therefore, processes started by **cron**, for example, as **sys** and that are run from **/usr/src**, prevent remote resources from being mounted on **/usr/src** because the directory is busy.

The administrator should select a different directory to be the current directory for the **sys** login in **/etc/passwd**. This will allow remote resources to be mounted on **/usr/src**.

PATH

The default **PATH** environment variable searches the current directory first. A superuser unknowingly may run a program in the current directory.

ports(1M)

The utility /etc/ports is an AT&T specific utility and is not supported. Use the utility /local/bin/mktty in place of /etc/ports.

prtconf(1M)

The utility prtconf is not supported in this release.

prtvtoc(1M)

The prtvtoc utility, run only by superuser, is used to print device specif information about disk devices, including disk dimensions, and logical di information such as disk type, starting sector, total sectors, mount flags, ar mount directory (if applicable). A "-v" option has been added to the prtvtoc ut ity which causes the logical disk types to be printed in a verbose format rath than just the number (i.e. SWAP rather than 2).

pump(1M)

The utility pump is an AT&T specific utility and is not supported.

rc2(1M)

The RMTMPFILES file, which is executed by **rc2**, now correctly remov all files in /usr/tmp when the system is initialized.

sadp(1M)

The utility sadp is not supported in this release.

setlp(1M)

If the **setlp** flag **NOCR** is set when using a parallel printer, the entire file now printed.

setlp(1M)

If formfeeds are set to 0 with **setlp** when using a parallel printer, formfeed is no longer printed at the end of the file.

setlp(1M)

Due to a new version of CG code, a double formfeed may occur wh using parallel printers. This may occur at the end of the page if the lines p page iss set to **l66**. To eliminate one of the formfeeds, set the lines per pa to **l0**.

shutdown(1M)

The previously unsupported **–g** and **–i** options of **shutdown** are implemented as described in the manual page.

singlepower

No longer shuts down the system in single-user state.

singlepower

Now calls **umountall** correctly.

strace(1M)

STREAMS problem has been corrected.

sysadm(1)

System administration partial subcommand names (for example, **for** for **format**) are not accepted from the shell. Do not use partial subcommand names.

sysadm(1)

The **sysadm** help file for setting a password does not warn of the 8 character limit on passwords. Someone who chooses a password "abcdefgh" is told that the password requires a numeric character, but the password is rejected when "abcdefgh9" is chosen because the number does not appear within the first 8 characters.

sysadm(1)

There is no protection provided against multiple users using the same **sysadm** subcommand at one time. This can cause problems when the **sysadm** subcommand references tape drives or system files.

sysadm(1)

The **sysadm** utilities that perform backup functions (**backup**, **store** and **drive**) prompt, as documented, with:

```
Mount the tape in the drive.
Press <RETURN> when ready.
```

sysadm(1)

A new sysadm menu, **sysadm tapemgmt mkrmt**, has been added. This option allows the user to create the new devices needed to use the 150 MB cartridge tape drive. If the user's system is equipped with a 150 MB cartridge tape drive rather than a 60 MB cartridge tape drive, the user must create new devices to use the drive. Devices for the 60 MB cartridge tape drive are the default devices (/dev/rmt1 and /dev/rmt0). The script **mkrmt** removes these defaults and creates the necessary devices.

sysadm adduser(1)

The **adduser** option of **sysadm** allows a new user's home directory to be a directory that already exists. The mode, owner and group of the existing directory will be given to the new user.

sysadm backup(1)

The **sysadm backup** procedure sometimes prints a failure message before it prints one saying that the procedure succeeded. In this case, ignore the message saying that the backup succeeded.

sysadm cpdisk(1)

The **cpdisk** utility accepts parameters and ensures the parameters are optimum for **dcopy** (**dcopy** is called by **cpdisk**).

sysadm datetime(1)

The menu **sysadm datetime** now allows the user to change the year past the turn of the century rather than setting an upper limit of 1999.

sysadm nodename(1)

The **nodename** utility no longer fails because the directory /etc/rc.d does not exist. The /etc/rc.d directory is supplied with the release.

sysadm portmgmt delete(1)

After executing **sysadm portmgmt delete**, the **inittab** entry for the port is not returned to a usable state. For example, if a modem is connected to tty21 and **portmgmt delete** is executed, the **/etc/inittab** file entry for tty21 looks similar to the following line:

```
21:2:respawn:/etc/getty -t 60 tty21 1200H
```

To change the entry to a usable state, log in as **root**, edit the **/etc/inittab** file, and change the entry for tty21 to look like the following line:

```
21:2:off:/etc/getty tty21 1200
```

sysadm portmgmt modify(1M)

If connecting a modem to a port that had a terminal connected to it, **sysadm portmgmt modify** may not start the **uugettys**. Before connecting a modem to the port, verify that a **getty** is running on the port. Execute the following command and look for the process number of the port to which the modem will be connected. The port can be identified by its tty number (for example, tty14 or tty22).

ps –ef

After the process number has been identified, execute the following command.

kill –9 *PID*

where *PID* is the process number. A modem can now be connected to the port. The **portmgmt modify** command should execute properly.

sysadm restore(1)

Previously, if **sysadm restore** was not given an absolute pathname when renaming files, the files would be saved in the root directory, and users were not told where they were. With ARIX-OS V.3, if files are renamed when restoring them with **sysadm filemgmt** or **sysadm restore**, and a complete pathname is not provided, the files are placed under **/usr/tmp** or **/tmp** and users are told where they are.

sysadm uucpmgmt(1)

sysadm uucpmgmt and **sysadm devicemgmt** use the last two digits of the **tty** name for the ID field in the **/etc/inittab** file. Thus, the same ID field is created for tty00 and tty100, which causes a problem with **init**. This problem occurs only when there are 7 or more GC boards. Users can solve this problem by editing **/etc/inittab** manually to change the first field.

sysadm uucpmgmt(1)

sysadm uucpmgmt states that users can enter a <CR> to select the default speed (console). If a <CR> is entered, an error message is returned saying that the default speed is not found in the **gettydefs** file. Instead of selecting the default speed, a baud rate needs to be entered; for example, 300, 1200, or 9600.

syssetup

nodename now passes correct parameters to **uname**.

tape utilities

The cartridge tape utilities, **ctccpio**, **ctcfmt**, and **ctcinfo** are not supported. The utility **cpio** can be used in place of **ctccpio**; **tension/erase** can be used rather than **ctcfmt**. The **ctcinfo** utility has no replacement.

ulim(1M)

Remote commands such as **rcp** and **rsh** now work on accounts with /local/bin/ulim as a login shell.

ulim(1M)

The user name can now be longer than eight characters.

/etc/ulimrc

The /etc/ulimrc file is now created when **ulim** is invoked.

umount(1M)

If attempting to **umount –d** a disk file system, the error message returned is as follows:

```
umount: /dev/dsk/c1dXXX not mounted
```

The **–d** option is valid only for remote **umount**s.

umountall(1M)

The **umountall** command now handles large file systems.

uname(1M)

When the command **uname -a** is entered, the report line now matches the documentation in the *ARIX–OS V.3 System Administrator's Guide*.

uucheck(1M)

uucheck without any options prints nothing; always use the **–v** option with **uucheck**.

Users cannot ask for different levels of debugging information with **uucheck –x**.

uucheck(1M), uucleanup(1M), Uutry(1M)

Most of the Basic Networking Utilities commands can be executed by users. The exceptions are **uucheck** and **uucleanup**, which require either an administrative (**uucp**) login or a **root** login.

uucheck, **uucleanup**, and **Uutry** are located in the **/usr/lib/uucp** directory, which is not in the search path for most logins, including those for **uucp** or **root**. Therefore, the full path name must be given, or be in the **/usr/lib/uucp** directory to execute these three commands.

Another alternative is to link the command where it may be easily accessed, for example, **/usr/bin**.

Uutry(1M)

When using the Basic Networking Utilities over a transport provider, and a remote system listens on an address different from that in the local **Systems** file, trying to **Uutry** to the remote system results in the following error message:

```
Connect failed: NO DEVICES AVAILABLE
```

This message does not necessarily imply that there are no available devices on the local system. However, it does mean that **Uutry** has failed after opening a device and before achieving a connection.

This failure could be caused by a variety of problems, including no devices available on the local system or the address in the **Systems** file being incorrect. To see the local devices that are in use, type **uustat –p**.

volcopy(1M)

Multiple volumes are now handled correctly.

volcopy(1M)

The user is now allowed to quit when a write error is encountered.

volcopy(1M)

The utility **volcopy** now properly asks for and sets up the tape information.

volcopy(1M)

volcopy will now allow the user to escape to the shell during a copy.

volcopy(1M)

Even though **volcopy –y** is supposed to answer all the questions **volcopy** asks affirmatively, users still have to reply to the first question. **–y** is an undocumented option.

:opy(1M)

volcopy prompts the user for several questions; however, if **volcopy** is run ıe background, the prompts do not appear on the screen even though **vol-y** is still waiting for the response from the terminal. Do not run **volcopy** in background.

:opy(1M)

The **volcopy** utility now supports 10000 bpi.

:opy(1M)

The **volcopy** utility miscalculates the number of tapes required if the "8000 ' option is selected.

l(1)

End-of-line characters have been cleaned up.

Programmer Commands, System Calls

misprints

In the *Programmer's Reference Manual*, some pages were incorrectly placed in the manual. The **shmctl**(2) entry is placed where **t_alloc**(3N) should be, and **t_alloc** is placed in Section 2 where **shmctl** should be. Also, the entries **getarg**(3F) and **getenv**(3F) were incorrectly inserted after the **plot**(3X) entry.

ld(1)

Link editor command files containing memory specifications no longer produce error messages.

lib

/lib/c2 is not a part of the new compiler and no longer exists.

libc.a

Instructions for profiling are no longer stripped out when using the optimizer.

libgen.a

The syntax error in the /usr/include/lib.h file has been corrected.

libgen.a

The script *mklib.h* has been modified to search for **sigset** and edit the file appropriately.

mkshlib(1)

The **mkshlib** utility now incorporates the **#hide** and **#export** directives.

sdb(1)

When running a program in **sdb**, register values were incorrect. The **sdb** utility now maps core files correctly, which allows stack traces to work properly and permits access to program variables when working from a core file.

acct(2)

The **acct** system call now sets **errno** to **EACCESS** when given an argument that is a directory instead of a file instead of setting it to **EISDIR** as before.

brk(2)

If allocating memory causes a deadlock, the **brk** system call fails, setting **errno** to **EAGAIN**.

In previous releases, it was possible for the system to become deadlocked if free swap space or free main memory were not available. When a deadlock occurred, users could not "swap in" a runnable process because there was no room in main memory, and it was also impossible to "swap out" a process to free the needed main memory because there was no room in the swap area. While rare, this situation occurred often enough to require adding checks in ARIX-OS V.3 to prevent it.

Several system calls now fail and set **errno** to indicate that a deadlock might have occurred. Figure 4-1 shows which system calls have changed and what value is given to **errno**.

System Call	**errno**
brk	**EAGAIN**
exec	**EAGAIN**
fork	**EAGAIN**
plock	**EAGAIN**
shmat	**ENOMEM**
shmctl	**ENOMEM**

Figure 4-1: Deadlock Detecting System Calls

Note that this new behavior should only occur when the system has nearly exhausted its memory capacity. If it occurs often, consider adding more main memory or increasing the size of the swap space.

exec(2)

If allocating memory causes a deadlock, the **exec** system call can fail setting **errno** to **EAGAIN**.

See discussion of deadlock detection in **brk**(2).

exec(2)

The **exec** system call now fails if the program to be run requires a shared library for which users do not have execute permission (**errno** set to **ELIBACC**), or if users try to exec a shared library directly (**errno** set to **ELIBEXEC**), or if users try to exec a shared library that does not exist.

fcntl(2)

The **fcntl** system call, on a file or record lock request, now sets **errno** to **ENOLCK** when the system runs out of lock resources instead of setting it to **ENOSPC** or **EMFILE** as before.

This change will only affect programs that currently check for **ENOSPC** or **EMFILE** to learn if the system has run out of lock resources.

Note that the effect will only be seen when the program is run and other programs have used up all the available locks.

fork(2)

When no unused entries remain in the system process table, the **fork** system call fails. No message is printed on the console to show that the system process table is full. When the **fork** call fails, **errno** is set to **EAGAIN**.

fork(2)

The **fork** system call now has additional reasons for failing, but it still sets **errno** to **EAGAIN** in these cases.

This should not affect a program s attempt to catch cases where the **fork** system call fails because the same **errno** value is used. Users should recognize, however, that the demand paging system introduces new ways for **fork** to fail when system resources are running low.

mount(2)

The **mount** system call now correctly fails, with **errno** set to **ENOTDIR**, on an attempt to mount a special file on itself.

mount(2)

The **mount** system call now fails and sets **errno** to **EINVAL** if the file system's type is not recognized or if the **mflag** (previously **rwflag**) argument is not correct.

open(2)

If **open** with flags (**O_RDONLY** | **O_CREAT**) is used on a directory, it is treated as if only **O_RDONLY** was set. If / is opened with (**O_RDONLY** | **O_CREAT**), the system call will fail with **errno** set to **ENOENT**.

plock(2)

If locking a process results in a memory deadlock, the **plock** system call fails setting **errno** to **EAGAIN**. See discussion of deadlock detection above (**brk**(2)).

ptrace(2)

The **ptrace** system call now allows write access to a shared text segment. This allows a program that more than one person may be running to be debugged.

In earlier releases, the **ptrace** system call refused to write into a text segment, or "pure procedure space," if that segment was being shared with another process and the other process was executing in the segment. The **ptrace** system call allows this write in ARIX-OS V.3 by ensuring that a separate text image is made and that the write access is to that separate image.

shmctl(2)

If locking the shared memory region causes a deadlock, the **shmctl** system call now fails setting **errno** to **ENOMEM**. For example, the following will fail if the attempt to lock the shared memory segment, identified by *shmid*, causes a deadlock:

shmctl(*shmid*, **SHM_LOCK)**

See the discussion of deadlock detection under the **brk**(2) section above.

shmctl(2)

The **shmctl** system call ignores an attempt to unlock a shared memory segment that is already unlocked instead of failing as before.

This change is not likely to cause a problem unless a program deliberately tries unlocking a shared memory segment without knowing if the segment is already locked in memory. If such a program looked for the **EINVAL** error return to indicate that the unlock attempt was not needed, it will no longer work as expected.

shmop(2)

If there is not enough memory to allocate page tables or if attaching the shared segment causes a deadlock, the **shmat** system call may fail setting **errno** to **ENOMEM**.

The unavailability of additional memory for tables needed to manage the separate pages of the shared segment, or the possibility of a memory deadlock, may also cause the system call to fail. For a discussion of deadlock detection, see **brk**(2) above.

shmop(2)

shmat system call now allows text as well as data segments to be shared.

Previously, a shared memory segment could only be attached to an address in the data segment of a process. With ARIX-OS V.3, a shared memory segment can be attached to any address in a process.

signal(2)

The **signal** system call now returns a pointer to a function of type **void** instead of a pointer to a function of type **int** as before.

This change was made to bring ARIX-OS V.3 closer to conforming with the IEEE standard. Since the function to which the return value of the **signal** system call points does not itself return a value, **void** is its correct type, not **int**.

No source code changes are required for ARIX-OS V.3, although continued use of the **int** type instead of the **void** type will cause a warning message from the **cc** compiler or the **lint** program checker. While this message is harmless and the code will compile correctly, start changing source code now to ensure compatibility with future ARIX-OS V.3 releases. All previously compiled programs and application packages that use the **signal** system call will still work with this release.

signal(2)

The signal **SIGIOT** is being phased out to be replaced with the signal **SIGABRT**. This change was made to bring ARIX-OS V.3 closer to conforming with the IEEE standard.

Currently, both names are supported so source code is compatible. In the future the name **SIGIOT** will no longer be supported, so start changing the source code now. However, the value of **SIGIOT** and the value of **SIGABRT** are the same, which means that all compiled programs, including application packages that may have been purchased, will continue to work, even in the future. For example, the **abort**(3C) library routine is now described as issuing the **SIGABRT** signal instead of the **SIGIOT** signal as before. New source code should be written to expect the **SIGABRT** signal. However, since the values are the same, a program previously compiled to expect the **SIGIOT** signal from **abort** will continue to work when linked with the new **abort** routine.

signal(2)

The **signal** system call may fail, setting **errno** to **EINVAL**, if the *func* argument is invalid.

Previously the **signal** system call did not check its second argument, *func*, to ensure that it was one of **SIG_DFL**, **SIG_IGN**, or a valid function address.

spell

A dictionary has been added.

uadmin(2)

The **uadmin** system call is AT&T specific and is not supported.

umount(2)

On an attempt to unmount a special device whose major and minor numbers do not exist, the **umount** system call now sets **errno** to **EINVAL** instead of **ENXIO**.

There should not be many programs affected by this change, since special devices are usually mounted and unmounted using the **mount**(1M) and **umount**(1M) shell commands. See if any of the programs that use the **umount**(2) system call check **errno** for the value **ENXIO** when the system call fails. Any that check for **ENXIO** should be changed to check for **EINVAL**.

umount(2)

For **umount**, **EBUSY** is now returned if the device of the file system to be unmounted is the default pipe device.

The default pipe device is that section of hard disk used for unamed pipes. If it is overridden and placed in a section of disk belonging to a file system that can be unmounted (for example, **/usr**), and a user attempts to unmount that file system, the unmount will fail with the above error.

unlink(2)

The **unlink** system call fails without issuing an error message when used to unlink a busy text file.

ustat(2)

A new error return has been added. If the root inode of the mounted file system that the user is doing the **ustat** on is NULL, **ENOENT** is set.

write(2)

The lock condition that occurred when **write** tried to access a block device has been corrected.

abort(3C)

The **abort** routine now issues the **SIGABRT** signal instead of the **SIGIOT** signal. See **signal**(2) in the section on system calls.

abort(3C)

The **abort** routine no longer closes files when the **SIGABRT** (previously **SIGIOT**) signal is being caught or ignored. Previously the **abort** routine would close all open files before issuing the **SIGIOT** signal that would normally cause the program to halt. If, however, the program had arranged to trap or ignore the **SIGIOT** signal, it would have to reopen the closed files before continuing.

With ARIX-OS V.3 the **abort** routine closes the files only if the program will halt on receiving the **SIGABRT** signal (which has the same value as the **SIGIOT** signal).

If a user has a program that used the **abort** routine and trapped or ignored the **SIGIOT** signal, the user should check to see if the new action by **abort** of keeping files open causes a problem.

ctime(3C)

The types of the argument **clock** in the **ctime**, **gmtime**, and **localtime** routines have been changed from "pointer to **long**" to "pointer to **time_t**." This change was made to bring ARIX-OS V.3 closer to conforming with the IEEE standard.

No source code changes are required for ARIX-OS V.3, but start changing source code now to ensure compatibility with future ARIX-OS V.3 releases. All previously compiled programs and application packages that use these routines will still work with this release.

curses(3X)

To facilitate debugging of **curses** applications, passing an invalid window pointer to a **curses** function causes a core dump. This problem can be circumvented by testing the return code of the function that created the window pointer.

curses(3X)

keypad, **meta**, **slk_clear**, and **slk_refresh** change the timestamp of the tty and can change the output of the **scr_dump**, **scr_init**, and **scr_restore** functions.

curses(3X)

Application programmers should not reference internal routines or members of structures. If they do, there may be compatibility problems with future releases of **curses**.

curses(3X)

cur_set no longer tries to simulate cursor modes that are undefined for the terminal being used.

curses(3X)

To maintain object mode compatibility with prior release of **libcurses**, all calls to **newterm** and **initscr** must appear in the **.o** that was compiled in the earliest release.

curses(3X)

In previous versions **meta**() was by window. With ARIX-OS V.3 it is by terminal and not by process. The change is necessary because the tty driver is by terminal and not by window.

curses(3X)

To use the new **curses** features, use the ARIX-OS V.3 version of **curses** on ARIX-OS V.3. All programs that ran with ARIX-OS 4.0 **curses** will also run on ARIX-OS V.3. Applications can be linked with object files based on ARIX-OS 4.0 **curses/terminfo** with the ARIX-OS V.3 **libcurses.a** library; however, applications cannot be linked with object files based on ARIX-OS V.3 **curses/terminfo** with the ARIX-OS 4.0 **libcurses.a** library.

curses(3X)

km (has_meta_key) is ignored. For **meta**() to work correctly in ARIX-OS 4.0, **km** had to be specified in the terminal's **terminfo** entry. With ARIX-OS V.3 and the change to **meta**(), **km** is no longer necessary.

curses(3X)

raw() (raw mode) no longer affects 8-bit mode. In ARIX-OS 4.0 calling **raw**() set 8-bit mode for the terminal.

curses(3X)

The **curses**(3X) manual page states that the effect of **slk_label** on blanks is that the label returned is in the same format as it was passed to **slk_set**(). The documentation in ARIX-OS 4.0 says that it stripped leading and trailing blanks.

curses(3X)

Until ARIX-OS V.3, if the **leaveok**() function was set to *TRUE*, **wrefresh**() left the cursor wherever it happened to be after the refresh was completed. In addition, the **getsyx**() function would return '**-1,-1**" if **leaveok**() was *TRUE*, and **setsyx**() would set **leaveok**() to *TRUE* if it was passed '**-1,-1**' as arguments. This behavior is correctly documented in the **curses**(3X) manual page.

In ARIX-OS V.3, however, when **leaveok**() is set to *TRUE*, **wrefresh**() moves the cursor to the last window that had **leaveok**() set to *FALSE*. Also, **getsyx**() does not return '**-1,-1**' if **leaveok**() is *TRUE*, and **setsyx**() will not set **leaveok**() to *TRUE* if it is passed '**-1,-1**'. Instead, **leaveok**() will remain at whatever state it happened to be at before the function call. The end result is that the cursor may end up in different positions with ARIX-OS V.3 **curses**, and some programs may produce unnecessary cursor movement.

curses(3X)

The declaration of **chtype** changed in ARIX-OS V.3 to an unsigned long from an unsigned short. It now has 16 attribute bits and 16 bits for data. The ARIX-OS 4.0 version of **libcurses.a** declared **chtype** as an unsigned short with 9 attribute bits and 7 data bits. However, object compatibility is preserved for applications that are re-linked with the ARIX-OS V.3 version of **/usr/lib/libcurses.a**. If a function *foo* required a 16 bit **chtype** argument, it will continue to have a 16 bit argument, and a new function *foo32* is created to accept a 32 bit **chtype** argument. Source compatibility is maintained by **#define**ing *foo* as *foo32* in **curses.h**.

Checking is not done on the values of pointers passed as arguments to some functions where previously the values were checked. These functions used to check the value and return with an error code if the pointer was null or out of bounds. This could cause an application that passes invalid pointer arguments to a **curses** function to core dump in ARIX-OS V.3 when it did not previously.

curses(3X)

The **box**() function has changed slightly from ARIX-OS 4.0. In ARIX-OS 4.0 the **box**() function used:

box (win, "-", "-")

and produced a solid line bordered window using the alternate character set. This produced the same results as:

box(win,0,0)

In ARIX-OS V.3, the previous example produces a window bordered by the characters "-" and "-".

curses(3X)

The routines **overlay**() and **overwrite**() have changed in ARIX-OS V.3. In ARIX-OS 4.0 the documentation stated that these routines overlayed all text from **scrwin**() on top of text from **dstwin**. In ARIX-OS V.3 the documentation correctly states that this occurs wherever the two windows overlap.

curses(3X)

In ARIX 4.0 the **_WCHAR** macro is defined incorrectly, and affects the following routines: **copywin**, **overlay**, and **overwrite**. These routines lose attributes in the copied region, while **box**() does not have its attributes set and **waddch**() and **winch**() also lose attributes.

Other routines may be affected because **waddch**() and **winch**() are called internally in many places. The effect will always be the same–lost attributes. There are no workarounds for this problem.

curses(3X)

In ARIX-OS V.3 **pechochar**() has two problems:

1. **pechochar**() does not recognize a **wmove**()
2. **pechochar**() does not update the cursor position

Use **waddch**() followed by **prefresh**() instead of **pechochar**().

fmod(3M)

According to the manual, **fmod**(*x*,*y*) should return *x* in the following cases:

y=0
x/*y* would cause an overflow

The value of *x* is now returned.

getopt(3C)

getopt is not supported in this release; **getopts** should be used in place of **getopt**.

libcurses(3X)

Because a declaration for an input/output in the libcurses routine **waddch** was not unsigned, the 8th bit was being stripped off. This problem has been corrected.

perror(3C)

When **perror** is called with an unknown error number, **perror** will now return the error number along with the "Unknown error" message.

puts(3S)

The **fputs** and **puts** routines now correctly return **EOF** if the attempt fails, instead of zero as before.

If there is a program that checks for a zero return from **puts** or **fputs** to indicate a write error, change it to check for **EOF**.

passwd(4)

When users try to change the password for a user who does not exist, **passwd** returns the following error message:

```
Permission denied.
```

This incorrect message appears even if **passwd** is executed as **root**.

terminfo(4)

If an 8-bit terminal needs an escape sequence that requires \0200, the sequence cannot be coded because \0200 in such a sequence is treated like a null character (\0).

terminfo(4)

The TERM name for wyse 50 terminals with the 132 column option are the normal wyse 50 names appended with a "w" (eg: wy50w). The following TERM strings are no longer supported:

4027-17ws	4027ex	aa
aaa-rv-unk	alto	bitgraph-ni
bg-ni	bitgraph-nv	bg2.0-nv
bg-nv	bg	bg2.0-rv
bg-rv	bitgraph-rv	bg1.25-nv
bg1.25-rv	blit-pb	ca
infoton	obitgraph	obitgraph-nv
obitgraph-rv	screwpoint	zeph

terminfo(4)

A **terminfo** description for the wy50w (Wyse 50 terminals, 132 column mode) has been added.

regexp.h(5)

regexp.h has been changed to allow for nested subexpressions.

If a regular expression has one or more occurrences of \(and no occurrences of \), the function **compile** will return with the statement **ERROR(42)** meaning \(, \) imbalance. Before, a check for matching parentheses was made only when a right parenthesis was encountered.

The change to allow nested subexpressions should not cause any problems. It is unlikely that someone would type in a regular expression with one or more instances of \(and no occurrences of \). Also, if one tried to reference an unbalanced subexpression with **number**, **grep** and other commands would complain with a "digit out of range" message. If one tried to reference an unbalanced subexpression in a substitution replacement pattern in **ed** and **sed**, a null string would be substituted for the subexpression.

Miscellany

Longest Allowed Pathnames

The longest pathname is now restricted to 512 bytes. System calls that require pathnames as arguments will now fail, setting **errno** to **ENOENT**, if a longer pathname is given.

Previously the pathname was not restricted by the operating system; however, most programs gave an ad hoc limit to the length. Generally these limits were well below 512 bytes, so most programs should not be affected by this change.

console

If running **shl** on the console, and then **shutdown –is** is run, the console hangs.

Exit **shl** before executing **shutdown**.

Converting to getopts by Hand

getoptcvt (see **getopts**(1)) adds about 30 lines of code to a shell script, so users may want to convert scripts by hand instead. Converting by hand probably will make the code cleaner and easier to understand. Also, users do not have to worry about parsing option-arguments that are also options.

Follow these guidelines to convert most scripts that currently use the **getopt**(1) command.

Step 1 Delete the old invocation line and the **if** statement that checks the exit code.

Step 2 Change the **for** loop to a **while** loop that invokes **getopt**(1).

Step 3 Change the patterns in the **case** statement from **–option** to single option letters.

Step 4 Delete the case for **—**.

Step 5 Add a case for **'?'**. This case may be used to print the usage message and to exit with a non-zero exit code. Note that the **?** is quoted since it is interpreted for filename expansion.

Step 6 Remove all **shift** commands within the **case** statement.

Step 7 Change **$2** to **$OPTARG** for cases that require an option argument.

Step 8 Add the statement **shift `expr $OPTIND – 1`** after the **while** loop so the remaining arguments can be referenced as before. Here is an example of a script before and after conversion:

```
# before conversion
set -- `getopt abo: $*`
if [ $? != 0 ]
then
        echo $USAGE
        exit 2
fi
for i in $*
do
        case $i in
        -a - -b)FLAG=$i; shift;;
        -o)OARG=$2; shift 2;;
        --)shift; break;;
        esac
done
```

```
# after conversion
while getopts abo: i
do
            case $i in
            a - b)FLAG=$i;;
            o)OARG=$OPTARG;;
            ?)echo $USAGE
            exit 2;;
            esac
done
shift `expr $OPTIND - 1`
```

To have the script work on releases before ARIX-OS V.3 Release 2.0 (that is, use either **getopts** or **getopt**), convert it as the example below shows:

```
if [ "$OPTIND" = 1 ]
then
        while getopts abo: i
        do
        case $i in
        a - b)FLAG=$i;;
        o)OARG=$OPTARG;;
        ?)echo $USAGE
        exit 2;;
        esac
        done
        shift `expr $OPTIND - 1`
        echo $*
else
        set -- `getopt abo: $*`
        if [ $? != 0 ]
        then
        echo $USAGE
        exit 2
        fi
        for i in $*
        do
        case $i in
        -a - -b)FLAG=$i; shift;;
        -o)OARG=$2; shift 2;;
        --)shift; break;;
        esac
        done
        echo $*
fi
```

Basic Networking Utilities (BNU): Intelligent Modems

Features have been added to the **/usr/lib/uucp/Dialers** and **/usr/lib/uucp/Devices** files to prevent problems that occur when using System 75s, System 85s, Hayes-compatible modems, and other intelligent modems that do not keep Carrier Detect (CD) high all the time.

Devices Adding a **,M** to the second field of an entry in the **Devices** file will cause the **O_NDELAY** flag to be set when the device is opened. This prevents BNU software from blocking on the device while waiting for CD. The example below shows how to add the **,M** to a **Devices** file entry for a device connected to an automatic call unit for a Hayes modem.

```
ACU tty11,M - 1200 hayes \T
```

Dialers Adding **\M** before the chat script in a **Dialers** file entry will set CLOCAL, preventing any change in the CD lead from resetting the state of the device. Once the conversation is established, **\m** will clear CLOCAL. This will allow BNU to again monitor changes in CD (for example, to notice if the line drops).

The example below shows how to add **\M** and **\m** to an entry for a Hayes modem in the **Dialers** file.

```
hayes  "=,-,"  ""  \M\dAT\r\c OK\r \EATDT\T\r\c CONNECT \m\c
```

NOTE

For some devices, adding a **\p** after the **\M** may be necessary.

/usr/lib/uucp/Devices

Comments in the **/usr/lib/uucp/Devices** file of the Basic Networking Utilities package show the protocol subfield attached to the fifth field of the **Devices** entry. This is incorrect. The protocol subfield should be attached to the first field of the entry. Verify the **Devices** to be sure the protocol subfield is attached to the first field.

The corrected comments are as follows:

```
(line 69)     #networkx,eg devicex - - TLIS \D
(line 81)     #                STARLAN,eg starlan - - TLIS \D
(line 87)     #networkx,eg devicex - - TLIS \D nls
(line 94)     #networkx,eg devicex - - TLI \D nls
```

Internationalization Notes

Below is a description of functionality differences that exist between ARIX-OS V.3 Release 2.0 international features and ARIX-OS 4.0. The changes for each component are described as well as any potential compatibility problems.

Functionality Changes

The following commands have functional differences in ARIX-OS V.3: **awk**, **cat**, **cpio**, **date**, **ed**, **egrep**, **find**, **grep**, **ls**, **make**, **mount**, **pg**, **pr**, **sed**, **sort**, and **vi**. These changes were made to implement 8-bit cleanup, date and time conventions, and character classification of different character sets. A few changes were also made to fix existing problems and for other enhancements. At the end of the section there is a list of new and changed environment variables.

Command Changes

cat(1)

The **cat** command was changed to remove its dependency on the ASCII code set when used with the **–v**, **–e** and **–t** options. The output of **cat** depends on the character classification table used. All printable characters are printed as the character itself. ASCII non-printable control characters continue to be printed as ^**C**. Non-printable 8-bit characters are printed as **M-x**, where **x** is the character specified by the low order 7 bits. The environment variable **CHRCLASS** is used to determine if a character is printable.

This change should not present compatibility problems because the functionality change only occurs when the **CHRCLASS** environment variable is set.

cpio(1)

The date output of the **–vt** option to **cpio** has changed depending on what the environment variable **LANGUAGE** is set to. In particular, the abbreviated month names are displayed in the appropriate language.

This change should not cause compatibility problems because the changed behavior occurs only when the **LANGUAGE** variable is set.

date(1)

The formatting software provided by the **date** utility has been moved to the **cftime** function in the C library, and **date** invokes the **cftime** routine with the appropriate arguments. **date** now supports date settings after the turn of the century and uses the environment variables **CFTIME** and **LANGUAGE** to define a default output format and to specify different languages.

These changes should not cause any compatibility problems if the internationalization features are not used because **date** functions differently only when the new environment variables are set or a format string has extra characters for a century setting.

ed(1)

ed no longer complains when a file has non-ASCII characters. The high-order bit is not stripped when reading in input and **ed** commands. If an encryption key is set, and the **CHRCLASS** environment variable is set to a non-null value and does not have the value **ascii**, **ed** now uses a different heuristic to determine if a file is encrypted. **ed**'s **l** command now prints octal codes for non-printable 8-bit characters. Also, **ed** now allows 8-bit characters in regular expressions. See the section on **regexp.h**, and the **–c** and **–x** options on the **ed**(1), **edit**(1), **ex**(1), and **vi**(1) manual pages in the *User's Reference Manual*.

The changed behavior occurs only when **ed** is used with 8-bit characters. A different heuristic to determine whether a file is encrypted is used only when the **CHRCLASS** environment variable is set.

egrep(1)

egrep now allows 8-bit characters in regular expressions and matches 8-bit characters in files correctly. Also, **egrep** uses the **CHRCLASS** environment variable to convert regular expressions and text to all lowercase for the **–i** option. This should not cause any problems because the functionality change occurs only when a new environment variable is set.

expr(1)

expr now processes 8-bit characters in regular expressions. These changes are described in detail in the section on **regexp.h**.

find(1)

find now processes 8-bit characters. A backslash is now used as an escape character, which allows users to escape the interpretation of metacharacters, for example,

```
find . -name '^\*' -print
```

prints all filenames that begin with a *. Previously, **find** would print all files whose names began with a \, and there was no way to escape the interpretation of metacharacters.

The use of a backslash as an escape character only causes problems in a shell script that uses **find** to print the names of files that have backslashes in them.

grep(1)

grep now processes 8-bit characters in regular expressions. These changes are described in detail in the section on **regexp.h**. **grep** also uses the **CHRCLASS** environment variable to convert regular expressions and text to all lowercase for the **–i** option. This change should not cause any problems because the functionality change occurs only when a new environment variable is set.

ls(1)

The date in the long **ls –l** format output will change based on the value of the environment variable **LANGUAGE**. In particular, the abbreviated month names are displayed in the appropriate language (see the section on **date**.)

When **ls** is used with the **–q** or **–b** options, the determination of whether a character is printable is based on the character class table identified by the environment variable **CHRCLASS**.

These changes should not cause any compatibility problems because the changed functionality occurs only when new environment variables are set or the environment variable **TZ** is unset or contains extra information for alternate time zones.

mount(1)

If **mount** is invoked with no arguments, the date output is based on the environment variable **LANGUAGE**. In particular, the abbreviated month and weekday names will be displayed in the appropriate language. The date and time output from **mount** is also affected differently by the **TZ** environment variable (see the section on **date**.)

These changes should not cause any compatibility problems because the functionality changes only when the **LANGUAGE** environment variable is set or the **TZ** environment variable is unset or contains extra information for alternate time zones.

pg(1)

pg now processes 8-bit characters in regular expressions (see the section on **regexp.h**). **pg** also correctly counts the number of printable characters on a line when it encounters 8-bit characters. This allows split lines that are longer than the screen width. The determination of whether a character is printable is based on the character class table identified by the environment variable **CHRCLASS**.

These functionality changes involve 8-bit characters and should not cause compatibility problems.

pr(1)

The date output of **pr** will change based on the environment variable **LANGUAGE**. In particular abbreviated month names will be displayed in the appropriate language. These changes should not cause any compatibility problems because the functionality changes only when the **LANGUAGE** environment variable is set or the **TZ** variable is unset or contains extra information for alternate time zones (see the section on **date**).

regexp.h

regexp.h is a header file used by **ed**, **expr**, **grep**, **pg**, and **sed**. It is also used by **bfs**, **nl**, **csplit**, and **acctcom**, but these commands are not part of ARIX-OS V.3 Release 2.0 international features. **regexp.h** was modified to allow 8-bit characters in character classes. Previously, when a character call was compiled, the high-order bit would be stripped from 8-bit characters.

The changes to **regexp.h** for 8-bit characters should not cause any problems.

sed(1)

sed now processes 8-bit characters in regular expressions (see the section on **regexp.h**) and **sed**'s **y** command. The **l** command now uses the environment variable **CHRCLASS** to determine if an 8-bit character is printable, and it will print an octal code if an 8-bit character is not printable. Previously **sed** would try to print an 8-bit character even if it was not printable.

These changes should not cause any compatibility problems because they only pertain to 8-bit characters.

sort(1)

The ASCII dependency of the **–d**, **–f**, and **–i** options to **sort** has been removed. The output now depends on the character classification table identified with the environment variable **CHRCLASS**. When the command is used with the **–M** option, the output depends on the value of the environment variable **LANGUAGE**.

The changes for the **–d**, **–f**, and **–i** options only pertain to 8-bit characters and should not cause any other compatibility problems. The change to the **–M** option takes effect only when the environment variable **LANGUAGE** is set and should not cause any compatibility problems.

vi(1)

vi no longer strips the high-order bit from 8-bit characters read in from text files, text insertion, and editing commands. It no longer looks for magic numbers of object files when reading in a text file. It also writes out text and displays text without stripping the high-order bit. Also, **–L** and **–r** options use the environment variables **LANGUAGE** and **TZ**.

vi now displays the octal codes of non-printable 8-bit characters in the text using the **CHRCLASS** environment variable to determine if a character is printable. It will also use the **CHRCLASS** environment variable to convert between upper-and lowercase characters for the tilde command and for the **ignorecase** option.

vi uses the same heuristic as **ed** to determine if a file is encrypted. The heuristic functions differently when the **CHRCLASS** environment variable is set (see the section on **ed**).

The change involving magic numbers should not cause any problems. The code looking for object file magic numbers is outdated because the object file header has changed. Also, **vi** looks for the magic numbers **0177555** and **0177545** which are only seen in old archive files. The new archive files have a string header. Also, the magic numbers contain **0377** in the high order byte. Except for null characters, **vi** can now display an object file, and there is no reason to forbid editing of object files.

The other changes should not cause any compatibility problems because they only occur for 8-bit characters or require that the **CHRCLASS** environment variable be set.

Environment Variables

CFTIME, **CHRCLASS**, and **LANGUAGE** are new environment variables in ARIX-OS V.3 Release 2.2. Setting them will cause the functionality of commands and C library functions to change. Also, the **TZ** environment variable may be interpreted slightly differently. The following table lists the commands and library functions that function differently with these variables.

Command/Function	Environment Variables
cat	**CHRCLASS**
cpio	**LANGUAGE, TZ**
date	**CFTIME, LANGUAGE, TZ**
ed	**CHRCLASS**
egrep	**CHRCLASS**
grep	**CHRCLASS**
ls	**CHRCLASS, LANGUAGE, TZ**
mount	**LANGUAGE, TZ**
nawk	**CHRCLASS**
pg	**CHRCLASS**
pr	**LANGUAGE, TZ**
sed	**CHRCLASS**
sort	**CHRCLASS, LANGUAGE**
vi,ex,edit	**CHRCLASS, LANGUAGE,TZ**

Kernel Notes

The following items describe the kernel modifications for ARIX-OS V.3 Release 2.2.

ANSI conformance

Files in **/usr/include** and **/usr/include/sys** have been modified to conform to ANSI specification for use with the new ARIX compiler products.

bdflushmax

This configurable variable allows **bdflush** to reduce the time delay caused when **bdflush** flushes large delayed write buffers..

buffer cache

The kernel has been modified to allow a much larger buffer cache to be configured than previous releases. The buffer cache may now exceed the 4MB kernel limit previously imposed.

System administrators may request a large buffer cache via the /usr/sys/cf/system file at configuration time. The configurable parameter "buffers" retains its meaning of "the total number of buffers in cache". A new tunable parameter **mbufmax** allows the driver compatibility buffer pool to be configured. The size of this pool cannot exceed 2200, and will always be allocated as the first part of the total buffer pool as described by "buffers". "buffers" should always be equal to or greater than **mbufmax**.

Note: Because of the 4MB kernel limit, the 2200 maximum may be less if other system tunables are changed.

For compatibility, block device drivers not provided by ARIX will be limited to the pool of compatibility buffers, unless they are modified to understand the kernel cache changes. Instructions to modify non-ARIX supplied drivers to utilize this feature is available through ARIX Customer Support.

bus errors

System bus mode errors that occur on slave CPUs are now reported on the master console.

bus errors

Minor fixes have been made to eliminate system mode bus errors. These errors typically occurred when the user configured many buffers, and then SCSI would attempt to use these buffers.

bus errors

Opening a 60MB cartridge device no longer causes a system mode bus error.

bus errors

The problem that caused occassional bus errors during heavy SCSI activity has been corrected.

constants

The values of some system constants have been changed to agree with the values in **/usr/include/limits.h**.

CPU speed

Users can request the speed of the CPU boards by using an ioctl call to any of the /dev/icb devices.

diagnostic code

If a kernel panic is caused due to an unrecognized bootimage version, the console displays the bootimage version number and other diagnostic information. Previously, only the message "improper config - kernel halted" was displayed.

disk error logging

Controller numbers are now included in the disk error log entries.

file size errors

Some core dumps may cause possible file size errors to be reported by **fsck**; these are only warnings and can be ignored. To determine whether the possible file size errors reported are resulting from core dumps, execute **ncheck –i** *i-number*, where *i-number* is given in the **fsck** message:

```
POSSIBLE FILE SIZE ERROR I=i-number
```

ncheck will generate the pathname of a file from its inode number, *i-number*. See the section "How To Check a File System for Consistency" in the *System Administrator's Guide* and **ncheck**(1M) in the *System Administrator's Reference Manual* for further details.

end-of-media messages

Abnormal "end-of-media" messages occurred when using cabled tape drives. This has been corrected.

GC tty mapping

The system dynamically allocates tty ports for GC boards. The system determines the number of tty ports on a board and allocates the correct number of device nodes. Previously, 16 nodes were allocated for all GC boards even though some boards had only 8 ports.

ieeefp.h

A define statement was added to **ieeefp.h** to enable user processes to be signaled if a floating point exception occurs.

kernel memory management

Problems resulting in erroneous memory fault addresses, EDAC errors and double bus faults have been corrected.

mnttab

The internal structure of **/etc/mnttab** has been modified to support NFS.

NFS file systems

A problem that prevented an NFS file system from being unmounted has been fixed.

non-RFS configured kernels

In non-RFS configured kernels, **ustat** did not properly return an error message when unreasonable device numbers were used; instead, the message `RFS not installed` was printed. This has been fixed.

RFS memory allocation

Memory has been reserved in the kernel so that if RFS is linked to the kernel, adequate space for RFS and other functions exists.

SCSI disk device driver

A SCSI disk device driver has been added to the standard kernel to support up to 56 disks on SCSI controllers.

SCSI disk device driver

The SCSI device driver has been modified to handle multiple SCSI EDT controllers correctly.

SCSI disk device driver

The setup for synchronous/asynchronous transfers has been improved.

SCSI disk device driver

Improper handling of rejected messages has been fixed.

SCSI disk device driver

The problem which caused a process to hang as a result of a timeout while attempting to send a read EOF command has been corrected.

SCSI disk device driver

The problem which caused a process to hang as a result of an illegal command returned by the tape drive when attempting to write a command after a read without an interceding return to BOT has been corrected.

SCSI disk device driver

The DMC does not recognize if there is a problem with the SCSI driver and will continue to look for data from the device, causing the system to hang. **check_dtb_status()** and **canceldmc()** now disable the current DMC process and tell it to start the next transfer.

slave CPU shutdown

Occasional shutdowns and/or faults of slave CPUs during heavy system usage have been corrected.

STREAMS

The kernel STREAMS handling has been upgraded to the ATT 5.3.2 level. It was previously a 5.3.1 implementation.

system calls

The semaphore, message, and shared memory system calls have been enhanced to run on all CPUs in a multiple processor environment.

user level profiling

A problem causing missing or incorrect user level profiling data, as reported to the user process, has been corrected.

write routine

The printer driver write routine allows printing of large blocks of data even when MUX, SNA, and BSC are implemented on the system.

X/OPEN library routines

The X/OPEN library routines use files in **/usr/include/sys**. The **dirent.h** file expected **/usr/include** to be used instead. The file **/usr/include/dirent.h** has been created and includes the **/usr/include/sys/dirent.h** file, which contains various structure definitions.

Bootimage Notes

This subsection contains information about changes to stand-alone utilities, the boot program, and disk controller download code.

disktest

The **disktest** program was changed to allow the user to proceed with the formatting of a disk even though the **disktest** program had problems determining the exact parameters of the disk. This could occur if the drive had not been formatted with the **disktest** program, or if certain sectors of a disk were unreadable.

disktest

The **disktest** program has been enhanced to support the 824 MB disk drives.

disktest

The **disktest** program now supports the SCSI EDT's. A maximum of 14 SCSI disks can be attached to each SCSI EDT. The drive number should consist of two digits (if necessary, pad with a leading 0) and be specified in hex.

disktest

Screen displays for SCSI controllers running **disktest** are different from the EDT controller displays. In particular, the Diagnostic Menu choice "g: display bad sector list" has been deleted.

dma

Canceling a chain dma results in an abort of the dma. Disable the chain dma before the abort.

dsetup

When invoking **dsetup** on SCSI drives, the drive number should consist of two digits (if necessary, pad with a leading 0) and be specified in hex.

dsetup

The available commands in the **dsetup** program have been altered to support SCSI controllers. The commands that have been deleted are the 'c' and 'u' commands from the "Bad Block Commands" section, 'b' from the "List Commands," and 'c' from the "Modify Commands" section. The 'p' option (List physical property) has been added to the "List Commands" section.

edac

The problem that caused abnormal edac during heavy usage of SCSI devices has been corrected.

EDT CRC errors

New micro code for EDT controllers has corrected the problem that caused CRC errors when using 824 MB Fuji drives.

EDT data transfer errors

Transfer overrun and alternate sector errors occurring simultaneously are now handled correctly on EDT controllers.

EDT disk requests

A problem of disk requests being lost with EDT controllers has been fixed.

EDT error reporting

Alternate sector and transfer overrun errors on EDT controllers are correctly reported.

EDT HEADER SEARCH errors

A problem causing HEADER SEARCH errors on swap devices has been eliminated.

EDT mixed drive types

The EDT controller now correctly handles multiple drive types on a single controller.

EDT raw/block errors

Errors with both raw and block I/O on the same disk drive would occasionally hang the system. This problem has been corrected.

EDT request buffers

EDT request buffers have been reserved for handling alternate sectors.

EDT tape retensioning

150 MB cartridge tape drives would sometimes hang during retensioning. This problem has been fixed.

EDT transfer overrun error recovery

Recovery time from transfer overrun errors has been substantially decreased.

EDT write protected disks

Problems that occurred when mounting and unmounting write protected disks have been corrected.

EOM errors

On nine-track tapes, **cpio** may hang if an EOM is encountered on a multi tape. An EOM error is now reported after the read of the last block is completed.

memory location

If more memory is required due to increased text and data size, move stack to highest memory location (that is, 0x3fff0).

system boot

The time involved in booting the system has been decreased.

150 MB cartridge tape drive

If the files on the tape are set up such that groups of files are separated by a file mark, reading through the first group of files using a device node that will not rewind on close will leave the tape positioned just after the file mark. The only legal 150 MB drive operations at this point are a read or read file mark. The 60 MB tape drive allows the user to write data at this point. The 150 MB drive would hang if a write is attempted. An illegal command status is now returned.

150 MB cartridge tape drive

The 150 MB cartridge tape drive exhibits symptoms which were not handled properly resulting in the drive hanging. Attempting to write to a non-qualified tape (e.g. 3M DC300XL) is considered illegal and will now return an illegal tape command error.

Writing to a unqualified tape will result in an "illegal command" error from the controller. This does not mean that the generic command was wrong, but only that the particular command is illegal when using the unqualified tape.

Known Deficiencies

The known deficiencies for ARIX-OS V.3 Release 2.2 follow:

cc(1)

The peephole optimizer will sometimes optimize out **asm(" ")** statements. It is recommended that routines containing **asm(" ")** statements be placed in a separate source module and be compiled without the **–O** option.

cc(1)

With **cc**, the following case:

```
struct {
                union {
                void            nothing;
                int             something;
                } un;
};
```

generates an inappropriate error message.

cc(1)

cc will clobber a file if the protection modes are set at 000.

cc(1)

If **cc** is invoked with the **–f** option, a message is produced stating that the **–f** option is unnecessary. As **cc** begins the link, it looks for the file /lib/fcrt1.o, which doesn't exist.

cflow(1)

A line in the file **/usr/bin/cflow** had a C-style comment line. The line has been changed to a shell comment line and **cflow** now executes successfully.

configuration files

During the remaking of the kernel after loading the overlay, it is possible that the dates on the system configuration files compared to the system file just loaded will be such that the configuration files will not be remade. This results in the error message

Unresolved symbol errors

during the loading portion of the kernel make. Should this occur, enter the following commands:

cd /usr/sys
rm cf/conf*

Then run the make again.

cpio(1)

Various **cpio** options are not performing as documented:

–I (uppercase i) When a file is created using **cpio -ov -Oxxx** and then is read back with **cpio -iv -Ixxx**, the message `N blocks` is produced, but no files are written to disk.

–k A usage message that the **–k** option is not available is produced.

–r The dot notation is understood to be "rename as . (pwd)," not as "keep that name."

–V The dots that should be displayed on the terminal as each file is copied are not produced until all the files are written.

csh(1)

When using a backquoted command in a *csh* shellscript (for example, 'pwd'), the value/string returned is sometimes null. For example,

```
set prompt="arix'pwd'$ "
alias cd 'cd :*; set prompt="alias'pwd'$ "'
```

ct(1), cu(1)

After completing a call with **ct** or **cu**, the phone line is not dropped at logout.

dd(1)

The utility **dd** throws away partial records with the command:

dd if=/dev/dsk/c0d0s4 of=/dev/null bs=256k

if the size of c0d0s4 is not an even multiple of 256K blocks.

disktest(1M)

The program **disktest** does not correctly determine the number of heads on a drive if there is a defect in sector 0 of a track. As a result, it is possible that a valid drive may be rejected.

dsetup(1M)

When the utility **dsetup** converts a cylinder/head/sector (chs) format hex number to a logical disk:block number sector (ld:blk) decimal format, the chs value is not checked to see whether it is out of range.

dsetup(1M)

An SMD drive cannot be selected after a SCSI drive with the **dsetup** utility. The error message `Not setup as a SCSI disc. Cannot continue.` is produced.

dsetup(1M)

The **c** option of **dsetup** asks for confirmation before continuing, even though no data is being written.

EDT timing value errors

CRC errors may cause changes in timing value results. The timing values for the drive in question need to be reset to the original values after CRC has finished.

fpgetround(3C)

The values given for FPU exception processing in the entry for **fpgetround** of the *ARIX-OS V.3 Programmer's Reference Manual* are incorrect. The include file should be consulted for the correct values.

fuser(1M)

The **/etc/fuser** utility works as documented on a device or disk slice, but fails when used on a file.

labelit(1M)

The utility **labelit** cannot be used to change the volume or pack number of a mounted file system.

labelit(1M)

A disk volume name with 6 characters produces garbled output when the volume name is printed.

ld(1)

User bus errors are generated by **ld** when a file table overflow occurs.

ldsa

The command **ldsa** implies that it is possible to transfer from any source to any destination. The transfer from a disk reserved area to tape results in an error message indicating the transfer requested too much data. The transfer from the reserved area of one disk to another also results in an error stating the image is too large to transfer.

mailx(1)

While in command mode with a list of mail messages to be read, **mailx** sometimes skips a message if a <return> is pressed at the prompt for next message. The message number must be specified at the prompt.

monitor proms

The **help** display for the monitor proms incorrectly reports that all commands except **e** and **g** should be followed by a carriage return. The **e** and **g** commands should also be followed by a carriage return.

SCSI drive

If an attempt is made to boot from a SCSI drive that has become detached from the system, the following non-interruptable error message continuously scrolls on the console: `disk access status: device check error`.

sectors

The operating system does not properly handle the problem of a bad sector in the available, "spare" sector list. If this occurs, then the file system should be of type "skip" so the sparing algorithm is not used. For more information, refer to the *System Administrator's Guide*.

sh(1)

While in the **sh**, if the TERM variable is not set and **vi**(1) is invoked, **vi** produces a message that an unknown terminal is being used. When a **Q** is entered, **vi** exits open mode and displays the **ex**(1) prompt (:). If the command to set the terminal type (for example, **set tty=wy50**) is entered followed by the command to invoke **vi** (**:vi**), **vi** produces the erroneous message that the tty does not have upline capability.

shutdown(1M)

The system cannot be reliably shutdown from an Ethernet port.

sysadm(1)

Several Top Menu **sysadm** options exhibit highlighting problems if a ^**A** (help option) is followed by ^**R** (previous menu or form). The options affected are:

- Compare Files
- Create, Copy, Rename, and Remove Files
- Directory-Related Commands
- Edit, Cut, and Paste Files
- Format Text and Hardcopy Output

sysadm checkfsys(1)

Prior to using **checkfsys** to check a file system, the file systems need to be labeled with **labelit**.

sysadm makefsys(1)

The **makefsys** script has a command line for choosing a disk slice. If a system has a large number of disk devices, the **exec**(2) system call's limitation of 5120 bytes in the argument list is exceeded and the **makefsys** script fails.

sysadm shutdown(1)

The **sysadm shutdown** utility uses the **shutdown 0** command instead of **shutdown -g0**

stty(1)

The settings for the **-raw** and **sane** commands do not correspond with their AT&T definitions. This problem is particularly apparent when switching from **raw** to *-raw*.

umount(1M)

If a disk slice is mounted without first running **mkfs**, the process hangs.

uname(1M)

When the command **uname -a** is entered, the report line does not match the documentation in the *ARIX-OS V.3 System Administrator's Guide*.

volcopy(1M)

The **volcopy** calculation for the number of tapes needed to complete the copy is incorrect by approximately twice the amount when the 150 MB cartridge drives are used.

Future Directions

awk, nawk, oawk

With ARIX-OS V.3 Release 2.2 there is a new **awk (nawk)** (see "Features of ARIX-OS V.3 Release 2.2" in these *Release Notes*). In the next major release of ARIX-OS V.3, **nawk** will be the default and will be linked to **awk** and **oawk** will refer to the Release 2.2 version of awk.

Regular Expressions

In ARIX-OS V.3 Release 2.2 the implementation of regular expressions (for example in **ex, egrep, regexp.h**, and **sh**) has been extended to support 8-bit characters. The semantics for the range notation, which currently uses ordinal values for the character, are the same as in previous releases. This permits ranges to include: 7-bit characters, 8-bit characters, and 7 and 8-bit characters.

In a future major release, ARIX-OS V.3 may support multiple character sets. The semantics for range expressions across character sets may change when support for multiple character sets is provided. Range expressions that contain 7 and 8-bit characters should be avoided due to this direction.

help and graph

In the next major release of ARIX-OS V.3, the **help** and **graph** utilities will be omitted from the release.

getdents(2)

The implementation of **getdents**(2) does not match the description in the **dirent**(4) manual page. The field **d_off** in **struct dirent** does not contain the file offset of the current directory entry but rather the file offset of the following entry. This will be corrected in the next major ARIX-OS V.3 release. The correction may require the re-compilation or re-linking of programs using the directory-management library routines described in **directory**(3X) (**opendir, closedir, readdir, telldir, seekdir, rewinddir**) and may require source changes to programs using the **getdents**(2) system call directly.

NAME

cc – C compiler

SYNOPSIS

cc [options] ... files

DESCRIPTION

The *cc* command is the interface to the C Compilation System. The compilation tools consist of a preprocessor, compiler, optimizer, assembler and link editor. The *cc* command processes the supplied options and then executes the various tools with the proper arguments. The *cc* command accepts several types of files as arguments:

Files whose names end with **.c** are taken to be C source programs and may be preprocessed, compiled, optimized, assembled and link edited. The compilation process may be stopped after the completion of any pass if the appropriate options are supplied. If the compilation process runs through the assembler then an object program is produced and is left in the file whose name is that of the source with **.o** substituted for **.c**. However, the **.o** file is normally deleted if a single C program is compiled and then immediately link edited. In the same way, files whose names end in .s are taken to be assembly source programs, and may be assembled and link edited; and files whose names end in .i are taken to be preprocessed C source programs and may be compiled, optimized, assembled and link edited. Files whose names do not end in **.c** , **.s** , or **.i** are handed to the link editor.

Since the *cc* command usually creates files in the current directory during the compilation process, it is necessary to run the *cc* command in a directory in which a file can be created.

The following options are interpreted by *cc*:

–B *string*
: Construct pathnames for substitute preprocessor, compiler, assembler, and link editor passes by concatenating *string* with suffixes **cpp, c0, coptim, c1, optim, as,** and **ld** . If *string* is empty, it is taken to be **/lib** for all programs except **as** and **ld**, for which it is **/bin.**

–c
: Suppress the link-editing phase of the compilation, and force an object file to be produced, even if only one program is compiled.

–D *symbol*
: Define *symbol* to the preprocessor. This mechanism is useful with the conditional statements in the preprocessor by allowing symbols to be defined external to the source file.

–E
: Run only *cpp*(1) on the named C programs, and send the result to the standard output.

–g
: Cause the compiler to generate additional information needed for the use of *sdb*(1). Using this option turns off optimization even if the **–O** option is specified.

–I *dir*
: Change the algorithm for searching for **#include** files whose names do not begin with / to look in *dir* before looking in the directories on

the standard list. Thus, **#include** files whose names are enclosed in double quotes are searched for first in the directory of the *file* argument, then in directories named in **-I** options, and last in directories on a standard list. For **#include** files whose names are enclosed in <>, the directory of the *file* argument is not searched.

-O Invoke the intermediate code and object code optimizers. Optimization cannot be used with the **-g** flag.

-o *outfile*
Produce an output object file by the name *outfile*. The name of the default file is **a.out**. This is a link editor option.

-P Run only *cpp*(1) on the named C programs, and leave the result on corresponding files suffixed with **.i**.

-p Arrange for the compiler to produce code which counts the number of times each routine is called; also, if link editing takes place, replace the standard startoff routine by one which automatically calls *monitor*(3C) at the start and arranges to write a **mon.out** file at normal termination of executions of the object program. An execution profile can then be generated by use of *prof*(1).

-Q When specified, **ld**(1) generates S90 binaries.

-S Compile the named C programs and leave the assembler-language output on corresponding files suffixed with **.s**.

-U *symbol*
Undefine *symbol* to the preprocessor.

-V Print the version number of the compiler.

W*c*,*arg1*[,*arg2*...]
Hand off the arguments[s] *argi* to pass *c*, where *c* is one of **[p0123al]** indicating preprocessor, compiler first pass, compiler second pass, intermediate code optimizer, assembler, or link editor, respectively. For example, **-Wa,-m** invokes the *m4*(1) macro preprocessor on the input to the assembler. This must be done for a source file that contains assembler escapes.

-w Suppress warning messages.

-Y[p0123alSILU],*dirname*
Specify a new pathname, *dirname*, for the locations of the tools and directories designated by the first argument. **[p0123alSILU]** represents:
p preprocessor
0 compiler pass 1 (c0)
1 compiler pass 2 (c1)
2 high level optimizer (coptim)
3 optimizer (optim)
a assembler
l loader (ld)

S directory that contains the startup routines
I default directory search by *cpp*(1)
L first default library searched by *ld*(1)
U second default library searched by *ld*(1)

–Z *date_stamp*
Set the date/time stamp in the object file. The format of the aurgument should be *mmddhhmmyy*. This option is passed to the assembler.

If the environment variable STALIGN is set to the value NO or is not set, then all structure elements that are larger than the type **char** are aligned on 16-bit boundaries. If STALIGN is set to YES then all structure elements larger than **short int** are aligned on 32-bit boundaries, as are any smaller types which are members of arrays. The types **short int** and **char** types that are not members of arrays remin on 16-bit and 8-bit boundaries, respectively.

FILES

file.c	input file
file.o	object file
file.s	assembly language file
a.out	link-edited file
/usr/tmp/mc68?	temporary
LIBDIR/cpp	preprocessor
LIBDIR/c0	compiler pass 1
LIBDIR/c1	compiler pass 2
LIBDIR/optim	peephole optimizer
LIBDIR/coptim	high-level optimizer
BINDIR/cc	compiler driver
BINDIR/as	assembler
BINDIR/ld	link editor *LIBDIR*/libc.a runtime library

SEE ALSO

as(1), ld(1).
The C Programming Language by B. W. Kernighan and D. M. Ritchie, Prentice-Hall, 1978.
Programming in C – A Tutorial by B. W. Kernighan.
C Reference Manual by D. M. Ritchie.
The C Programming Language in the Software Generation System Guide.

DIAGNOSTICS

The diagnostics produced by the C compiler are sometimes cryptic. Occasional messages may be produced by the assembler or link editor.

AME

cpio – copy file archives in and out

YNOPSIS

cpio –o [**acBvV**] [**–C** *bufsize*] [[**–O** *file*] [**–M** *message*]]

cpio –i [**ABcdmrtuvVfsSbk**] [**–C** *bufsize*] [[**–I** *file*] [**–M** *message*]] [*pattern* ...]

cpio –p [**adlmuvV**] directory

ESCRIPTION

cpio –o (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information. Output is padded to a 512-byte boundary by default.

cpio –i (copy in) extracts files from the standard input, which is assumed to be the product of a previous **cpio –o**. Only files with names that match *patterns* are selected. *patterns* are regular expressions given in the filename-generating notation of *sh*(1). In *patterns*, meta-characters ?, *, and [...] match the slash (/) character, and backslash (\) is an escape character. A ! meta-character means *not*. (For example, the **!abc*** pattern would exclude all files that begin with **abc**.) Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is * (i.e., select all files). Each *pattern* must be enclosed in double quotes otherwise the name of a file in the current directory is used. Extracted files are conditionally created and copied into the current directory tree based upon the options described below. The permissions of the files will be those of the previous **cpio –o**. The owner and group of the files will be that of the current user unless the user is super-user, which causes *cpio* to retain the owner and group of the files of the previous **cpio –o**. NOTE: If **cpio –i** tries to create a file that already exists and the existing file is the same age or newer, *cpio* will output a warning message and not replace the file. (The **–u** option can be used to unconditionally overwrite the existing file.)

cpio –p (pass) reads the standard input to obtain a list of path names of files that are conditionally created and copied into the destination *directory* tree based upon the options described below.

The meanings of the available options are

–a Reset *access* times of input files after they have been copied. Access times are not reset for linked files when **cpio –pla** is specified.

–A Allows a user to read archive tapes made on a foreign system, and tapes made by doing a **cpio** to a disk file, and then using **/bin/dd** to copy this file to archive tape.

–b Reverse the order of the *bytes* within each word. Use only with the **–i** option.

–B Input/output is to be blocked 5,120 bytes to the record. The default buffer size is 512 bytes when this and the C options are not used. (**–B** does not apply to the *pass* option; **–B** is meaningful only with data directed to or from a character special device, e.g. **/dev/rmt1**.)

–c Write header information in ASCII *character* form for portability. Always use this option when origin and destination machines are different types.

–C *bufsize*
Input/output is to be blocked *bufsize* bytes to the record, where *bufsize* is replaced by a positive integer. The default buffer size is 512 bytes when this and **B** options are not used. (**–C** does not apply to the *pass* option; **–C** is meaningful only with data directed to or from a character special device, e.g. **/dev/rmt1**.)

–d *directories* are to be created as needed.

–f Copy in all *files* except those in *patterns*. (See the paragraph on **cpio –i** for a description of *patterns*.)

–I *file* Read the contents of *file* as input. If *file* is a character special device, when the first medium is full replace the medium and type a carriage return to continue to the next medium. Use only with the **-i** option.

–k Attempt to skip corrupted file headers and I/O errors that may be encountered. If you want to copy files from a medium that is corrupted or out of sequence, this option lets you read only those files with good headers. (For *cpio* archives that contain other *cpio* archives, if an error is encountered *cpio* may terminate prematurely. *cpio* will find the next good header, which may be one for a smaller archive, and terminate when the smaller archive's trailer is encountered.) Used only with the **-i** option.

–l Whenever possible, *link* files rather than copying them. Usable only with the **–p** option.

–m Retain previous file *modification* time. This option is ineffective on directories that are being copied.

–M *message*
Define a message to use when switching media. When you use the **–O** or **–I** options and specify a character special device, you can use this option to define the message that is printed when you reach the end of the medium. One **%d** can be placed in the message to print the sequence number of the next medium needed to continue.

–O *file* Direct the output of cpio to *file*. If *file* is a character special device, when the first medium is full replace the medium and type a carriage return to continue to the next medium. Use only with the **–o** option.

–r Interactively *rename* files. If the user types a null line, the file is skipped. If the user types a "." the original pathname will be copied. (Not available with **cpio –p**.)

–s *swap* bytes within each half word. Use only with the **–i** option.

–S *Swap* halfwords within each word. Use only with the **–i** option.

–t Print a *table of contents* of the input. No files are created.

–u Copy *unconditionally* (normally, an older file will not replace a newer file with the same name).

–v *verbose*: causes a list of file names to be printed. When used with the **–t** option, the table of contents looks like the output of an **ls –l** command (see **ls**(1)).

–V *Special*Verbose: print a dot for each file seen. Useful to assure the user that **cpio** is working without printing out all file names.

NOTE: *cpio* assumes four-byte words.

If *cpio* reaches end of medium (end of a disk for example), when writing to (**–o**) or reading from (**–i**) a character special device, and **–O** and **–I** aren't used, *cpio* will print the message:

If you want to go on, type device/file name when ready.

To continue, you must replace the medium and type the character special device name (**/dev/rmt1** for example) and carriage return. You may want to continue by directing *cpio* to use a different device. For example, if you have two 9-track tape drives you may want to switch between them so *cpio* can proceed while you are changing the tapes. (A carriage return alone causes the *cpio* process to exit.)

EXAMPLES

The following examples show three uses of *cpio*.

When standard input is directed through a pipe to **cpio –o**, it groups the files so they can be directed (>) to a single file (**../newfile**). The **c** option insures that the file will be portable to other machines. Instead of *ls*(1), you could use *find*(1), *echo*(1), *cat*(1), etc. to pipe a list of names to *cpio*. You could direct the output to a device instead of a file.

ls | cpio –oc > ../*newfile*

cpio –i uses the output file of **cpio –o** (directed through a pipe with **cat** in the example), extracts those files that match the patterns (**memo/a1**, **memo/b***), creates directories below the current directory as needed (**–d** option), and places the files in the appropriate directories. The **c** option is used when the file is created with a portable header. If no patterns were given, all files from *newfile* would be placed in the directory.

cat newfile | cpio –icd "*memo/a1*" "*memo/b**"

cpio –p takes the file names piped to it and copies or links (**–l** option) those files to another directory on your machine (*newdir* in the example). The **–d** options says to create directories as needed. The **–m** option says retain the modification time. (It is important to use the **–depth** option of *find*(1) to generate path names for *cpio*. This eliminates problems *cpio* could have trying to create files under read-only directories.)

find . –depth –print | cpio –pdlmv *newdir*

SEE ALSO

ar(1), cat(1), echo(1), find(1), ls(1), tar(1).
cpio(4) in the *System Administrator's Reference Manual*.

NOTES

1) Path names are restricted to 256 characters.
2) Only the super-user can copy special files.
3) Blocks are reported in 512-byte quantities.
4) If a file has 000 permissions, contains more than 0 characters of data, and the user is not root, the file will not be saved or restored.

NAME

csh – a shell (command interpreter) with C-like syntax

SYNOPSIS

csh [**–cefinstvVxX**] [arg ...]

DESCRIPTION

Csh is a first implementation of a command language interpreter incorporating a history mechanism (see **History Substitutions**) and a C-like syntax.

NOTE: Job control facilities, file name completion, and timing/resource monitoring are not supported in this release of **csh**. See CAVEATS for more details.

An instance of *csh* begins by executing commands from the file '.cshrc' in the *home* directory of the invoker. If this is a login shell then it also executes commands from the file '.login' there. It is typical for users to put the command "stty erase '^h' kill '^u' echoe" in their *.login* file.

In the normal case, the shell will then begin reading commands from the terminal, prompting with '% '. Processing of arguments and the use of the shell to process files containing command scripts will be described later.

The shell then repeatedly performs the following actions: a line of command input is read and broken into *words*. This sequence of words is placed on the command history list and then parsed. Finally each command in the current line is executed.

When a login shell terminates it executes commands from the file '.logout' in the users home directory.

Lexical structure

The shell splits input lines into words at blanks and tabs with the following exceptions. The characters '&' '|' ';' '<' '>' '(' ')' form separate words. If doubled in '&&', '||', '<<' or '>>' these pairs form single words. These parser metacharacters may be made part of other words, or prevented their special meaning, by preceding them with '\'. A newline preceded by a '\' is equivalent to a blank.

In addition strings enclosed in matched pairs of quotations, '', '^' or '"", form parts of a word; metacharacters in these strings, including blanks and tabs, do not form separate words. These quotations have semantics to be described subsequently. Within pairs of '' or "" characters a newline preceded by a '\' gives a true newline character.

When the shell's input is not a terminal, the character '#' introduces a comment which continues to the end of the input line. It is prevented this special meaning when preceded by '\' and in quotations using '^', '', and "".

Commands

A simple command is a sequence of words, the first of which specifies the command to be executed. A simple command or a sequence of simple commands separated by '|' characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next. Sequences of

pipelines may be separated by ';', and are then executed sequentially. A sequence of pipelines may be executed without immediately waiting for it to terminate by following it with an '&'.

Any of the above may be placed in '(' ')' to form a simple command (which may be a component of a pipeline, etc.) It is also possible to separate pipelines with '||' or '&&' indicating, as in the C language, that the second is to be executed only if the first fails or succeeds respectively. (See *Expressions.*)

Jobs

The 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 asynchronously with '&', the shell prints a line which looks like:

[1] 1234

indicating that the job which was started asynchronously was job number 1 and had one (top-level) process, whose process id was 1234.

A job being run in the background will take its input from the empty file, */dev/null*, if no other input is provided. Background jobs are allowed to produce output.

Status reporting

This shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work. If, however, you set the shell variable *notify*, the shell will notify you immediately of changes of status in background jobs. There is also a shell command *notify* which marks a single process so that its status changes will be immediately reported. By default *notify* marks the current process; simply say 'notify' after starting a background job to mark it.

Substitutions

We now describe the various transformations the shell performs on the input in the order in which they occur.

History substitutions

History substitutions place words from previous command input as portions of new commands, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing and a high degree of confidence. History substitutions begin with the character '!' and may begin **anywhere** in the input stream (with the proviso that they **do not** nest.) This '!' may be preceded by an '\' to prevent its special meaning; for convenience, a '!' is passed unchanged when it is followed by a blank, tab, newline, '=' or '('. (History substitutions also occur when an input line begins with '^'. This special abbreviation will be described later.) Any input line which contains history substitution is echoed on the terminal before it is executed as it could have been typed without history substitution.

Commands input from the terminal which consist of one or more words are saved on the history list. The history substitutions reintroduce sequences of words from these saved commands into the input stream. The size of which is controlled by the *history* variable; the previous command is always retained, regardless of its value. Commands are numbered sequentially from 1.

For definiteness, consider the following output from the *history* command:

```
 9 write michael
10 ex write.c
11 cat oldwrite.c
12 diff *write.c
```

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but the current event number can be made part of the *prompt* by placing an '!' in the prompt string.

With the current event 13 we can refer to previous events by event number '!11', relatively as in '!-2' (referring to the same event), by a prefix of a command word as in '!d' for event 12 or '!wri' for event 9, or by a string contained in a word in the command as in '!?mic?' also referring to event 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; thus '!!' alone is essentially a *redo*.

To select words from an event we can follow the event specification by a ':' and a designator for the desired words. The words of an input line are numbered from 0, the first (usually command) word being 0, the second word (first argument) being 1, etc. The basic word designators are:

0	first (command) word
n	n'th argument
↑	first argument, i.e. '1'
$	last argument
%	word matched by (immediately preceding) ?s? search
x-y	range of words
-y	abbreviates '0-y'
*	abbreviates '↑-$', or nothing if only 1 word in event
x*	abbreviates 'x-$'
x-	like 'x*' but omitting word '$'

The ':' separating the event specification from the word designator can be omitted if the argument selector begins with a '↑', '$', '*' '-' or '%'. After the optional word designator can be placed a sequence of modifiers, each preceded by a ':'. The following modifiers are defined:

h	Remove a trailing pathname component, leaving the head.
r	Remove a trailing '.xxx' component, leaving the root name.
e	Remove all but the extension '.xxx' part.
s/l/r/	Substitute l for r
t	Remove all leading pathname components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, prefixing the above, e.g. 'g&'.

p	Print the new command but do not execute it.
q	Quote the substituted words, preventing further substitution.
x	Like q, but break into words at blanks, tabs and newlines.

(NOTE: The word designator is required when using the "e" or "t" modifiers.)

Unless preceded by a 'g' the modification is applied only to the first modifiable word. With substitutions, it is an error for no word to be applicable.

The left hand side of substitutions are not regular expressions in the sense of the editors, but rather strings. Any character may be used as the delimiter in place of '/'; a '\' quotes the delimiter into the *l* and *r* strings. The character '&' in the right hand side is replaced by the text from the left. A '\' quotes '&' also. A null *l* uses the previous string either from a *l* or from a contextual scan string *s* in '!?*s*?'. **The trailing delimiter in the substitution may be omitted if a newline follows immediately as may the trailing '?' in a contextual scan.**

A history reference may be given without an event specification, e.g. '!$'. In this case the reference is to the previous command unless a previous history reference occurred on the same line in which case this form repeats the previous reference. Thus '!?foo?↑ !$' gives the first and last arguments from the command matching '?foo?'.

A special abbreviation of a history reference occurs when the first non-blank character of an input line is a '↑'. This is equivalent to '!:s↑' providing a convenient shorthand for substitutions on the text of the previous line. Thus '↑lb↑lib' fixes the spelling of 'lib' in the previous command. Finally, a history substitution may be surrounded with '{' and '}' if necessary to insulate it from the characters which follow. Thus, after 'ls –ld ~paul' we might do '!{l}a' to do 'ls –ld ~paula', while '!la' would look for a command starting 'la'.

Quotations with ' and "

The quotation of strings by '`'` and '`"`' can be used to prevent all or some of the remaining substitutions. Strings enclosed in '`'`' are prevented any further interpretation. Strings enclosed in '`"`' may be expanded as described below.

In both cases the resulting text becomes (all or part of) a single word; only in one special case (see *Command Substitition* below) does a '`"`' quoted string yield parts of more than one word; '`'`' quoted strings never do.

Alias substitution

The shell maintains a list of aliases which can be established, displayed and modified by the *alias* and *unalias* commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, then the text which is the alias for that command is reread with the history mechanism available as though that command were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, then the argument list is left unchanged.

Thus if the alias for 'ls' is 'ls –l' the command 'ls /usr' would map to 'ls –l /usr', the argument list here being undisturbed. Similarly if the alias for 'lookup'

was 'grep !* /etc/passwd' then 'lookup bill' would map to 'grep bill /etc/passwd'.

If an alias is found, the word transformation of the input text is performed and the aliasing process begins again on the reformed input line. Looping is prevented if the first word of the new text is the same as the old by flagging it to prevent further aliasing. Other loops are detected and cause an error.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus we can 'alias print 'pr \!* | lpr'' to make a command which *pr's* its arguments to the line printer.

Variable substitution

The shell maintains a set of variables, each of which has as value a list of zero or more words. Some of these variables are set by the shell or referred to by it. For instance, the *argv* variable is an image of the shell's argument list, and words of this variable's value are referred to in special ways.

The values of variables may be displayed and changed by using the *set* and *unset* commands. Of the variables referred to by the shell a number are toggles; the shell does not care what their value is, only whether they are set or not. For instance, the *verbose* variable is a toggle which causes command input to be echoed. The setting of this variable results from the **-v** command line option.

Other operations treat variables numerically. The '@' command permits numeric calculations to be performed and the result assigned to a variable. Variable values are, however, always represented as (zero or more) strings. For the purposes of numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After the input line is aliased and parsed, and before each command is executed, variable substitution is performed keyed by '$' characters. This expansion can be prevented by preceding the '$' with a '\' except within "'s where it **always** occurs, and within `'`'s where it **never** occurs. Strings quoted by `` ` `` are interpreted later (see *Command substitution* below) so '$' substitution does not occur there until later, if at all. A '$' is passed unchanged if followed by a blank, tab, or end-of-line.

Input/output redirections are recognized before variable expansion, and are variable expanded separately. Otherwise, the command name and entire argument list are expanded together. It is thus possible for the first (command) word to this point to generate more than one word, the first of which becomes the command name, and the rest of which become arguments.

Unless enclosed in "" or given the ':q' modifier the results of variable substitution may eventually be command and filename substituted. Within "", a variable whose value consists of multiple words expands to a (portion of) a single word, with the words of the variables value separated by blanks. When the ':q' modifier is applied to a substitution the variable will expand to multiple words with each word separated by a blank and quoted to prevent later command or filename substitution.

The following metasequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable which is not set.

$name
${name}
Are replaced by the words of the value of variable *name*, each separated by a blank. Braces insulate *name* from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character is considered a letter.
If *name* is not a shell variable, but is set in the environment, then that value is returned (but **: modifiers and the other forms given below are not available in this case).**

$name[selector]
${name[selector]}
May be used to select only some of the words from the value of *name*. The selector is subjected to '$' substitution and may consist of a single number or two numbers separated by a '–'. The first word of a variable's value is numbered '1'. If the first number of a range is omitted it defaults to '1'. If the last member of a range is omitted it defaults to '$#name'. The selector '*' selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

$#name
${#name}
Gives the number of words in the variable. This is useful for later use in a '[selector]'.

$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 modifiers ':h', ':t', ':r', ':q' and ':x' may be applied to the substitutions above as may ':gh', ':gt' and ':gr'. If braces '{' '}' appear in the command form then the modifiers must appear within the braces. **The current implementation allows only one ':' modifier on each '$' expansion.**

The following substitutions may not be modified with ':' modifiers.

$?name
${?name}
Substitutes the string '1' if name is set, '0' if it is not.

$?0
Substitutes '1' if the current input filename is known, '0' if it is not.

$$

Substitute the (decimal) process number of the (parent) shell.

$<

Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a shell script.

Command and filename substitution

The remaining substitutions, command and filename substitution, are applied selectively to the arguments of builtin commands. This means that portions of expressions which are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

Command substitution

Command substitution is indicated by a command enclosed in '`'. The output from such a command is normally broken into separate words at blanks, tabs and newlines, with null words being discarded, this text then replacing the original string. Within '"'s, only newlines force new words; blanks and tabs are preserved.

In any case, the single final newline does not force a new word. Note that it is thus possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

Filename substitution

If a word contains any of the characters '*', '?', '[' or '{' or begins with the character '~', then that word is a candidate for filename substitution, also known as 'globbing'. This word is then regarded as a pattern, and replaced with an alphabetically sorted list of file names which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing file name, but it is not required for each pattern to match. Only the metacharacters '*', '?' and '[' imply pattern matching, the characters '~' and '{' being more akin to abbreviations.

In matching filenames, the character '.' at the beginning of a filename or immediately following a '/', as well as the character '/' must be matched explicitly. The character '*' matches any string of characters, including the null string. The character '?' matches any single character. The sequence '[...]' matches any one of the characters enclosed. Within '[...]', a pair of characters separated by '-' matches any character lexically between the two.

The character '~' at the beginning of a filename is used to refer to home directories. Standing alone, i.e. '~' it expands to the invokers home directory as reflected in the value of the variable *home*. When followed by a name consisting of letters, digits and '-' characters the shell searches for a user with that name and substitutes their home directory; thus '~ken' might expand to '/usr/ken' and '~ken/chmach' to '/usr/ken/chmach'. If the character '~' is followed by a character other than a letter or '/' or appears not at the beginning of a word, it is left undisturbed.

The metanotation 'a{b,c,d}e' is a shorthand for 'abe ace ade'. Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. Thus '~source/s1/{oldls,ls}.c' expands to '~usr/source/s1/oldls.c ~/usr/source/s1/ls.c' whether or not these files exist without any chance of error if the home directory for 'source' is '~/usr/source'. Similarly '../{memo,*box}' might expand to '../memo ../box ../mbox'. (Note that 'memo' was not sorted with the results of matching '*box'.) As a special case '{', '}' and '{}' are passed undisturbed.

Input/output

The standard input and standard output of a command may be redirected with the following syntax:

< name
: Open file *name* (which is first variable, command and filename expanded) as the standard input.

<< word
: Read the shell input up to a line which is identical to *word*. *Word* is not subjected to variable, filename or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting '\', '"', '' or '`' appears in *word* variable and command substitution is performed on the intervening lines, allowing '\' to quote '$', '\' and '`'. Commands which are substituted have all blanks, tabs, and newlines preserved, except for the final newline which is dropped. The resultant text is placed in an anonymous temporary file which is given to the command as standard input.

> name
>! name
>& name
>&! name
: The file *name* is used as standard output. If the file does not exist then it is created; if the file exists, its is truncated, its previous contents being lost.

  If the variable *noclobber* is set, then the file must not exist or be a character special file (e.g. a terminal or '/dev/null') or an error results. This helps prevent accidental destruction of files. In this case the '!' forms can be used and suppress this check.

  The forms involving '&' route the diagnostic output into the specified file as well as the standard output. *Name* is expanded in the same way as '<' input filenames are.

>> name
>>& name
>>! name
>>&! name
: Uses file *name* as standard output like '>' but places output at the end of the file. If the variable *noclobber* is set, then it is an error for the file not to exist unless one of the '!' forms is given. Otherwise similar to '>'.

A command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather they receive the original standard input of the shell. The '<<' mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input. Note that the default standard input for a command run detached is the empty file */dev/null*.

Diagnostic output may be directed through a pipe with the standard output. Simply use the form '|&' rather than just '|'.

Expressions

A number of the builtin commands (to be described subsequently) take expressions, in which the operators are similar to those of C, with the same precedence. These expressions appear in the *@*, *exit*, *if*, and *while* commands. The following operators are available:

| || &&& | ^ & == != =~ !~ <= >= < > << >> + - * / %
! ~ ( )

Here the precedence increases to the right, '==' '!=' '=~' and '!~', '<=' '>=' '<' and '>', '<<' and '>>', '+' and '-', '*' '/' and '%' being, in groups, at the same level. The '==' '!=' '=~' and '!~' operators compare their arguments as strings; all others operate on numbers. The operators '=~' and '!~' are like '!=' and '==' except that the right hand side is a *pattern* (containing, e.g. '*'s, '?'s and instances of '[...]') against which the left hand operand is matched. This reduces the need for use of the *switch* statement in shell scripts when all that is really needed is pattern matching.

Strings which begin with '0' are considered octal numbers. Null or missing arguments are considered '0'. The result of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word; except when adjacent to components of expressions which are syntactically significant to the parser ('&' '|' '<' '>' '(' ')') they should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in '{' and '}' and file enquiries of the form '-*l* **name**' **where** *l* is one of:

r	read access
w	write access
x	execute access
e	existence
o	ownership
z	zero size
f	plain file
d	directory

The specified name is command and filename expanded and then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible then all enquiries return false, i.e. '0'. Command executions

succeed, returning true, i.e. '1', if the command exits with status 0, otherwise they fail, returning false, i.e. '0'. If more detailed status information is required then the command should be executed outside of an expression and the variable *status* examined.

Control flow

The shell contains a number of commands which can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict the placement of some of the commands.

The *foreach*, *switch*, and *while* statements, as well as the *if-then-else* form of the *if* statement require that the major keywords appear in a single simple command on an input line as shown below.

If the shell's input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward goto's will succeed on non-seekable inputs.)

Builtin commands

Builtin commands are executed within the shell. If a builtin command occurs as any component of a pipeline except the last then it is executed in a subshell.

alias
alias name
alias name wordlist
The first form prints all aliases. The second form prints the alias for name. The final form assigns the specified *wordlist* as the alias of *name*; *wordlist* is command and filename substituted. *Name* is not allowed to be *alias* or *unalias*.

alloc
Shows the amount of dynamic memory acquired, broken down into used and free memory. With an argument shows the number of free and used blocks in each size category. The categories start at size 8 and double at each step. This command's output may vary across system types, since systems may use a different memory allocator. NOTE: **alloc** is not available in this release.

break
Causes execution to resume after the *end* of the nearest enclosing *foreach* or *while*. The remaining commands on the current line are executed. Multi-level breaks are thus possible by writing them all on one line.

breaksw
Causes a break from a *switch*, resuming after the *endsw*.

case label:
A label in a *switch* statement as discussed below.

cd
cd name
chdir
chdir name

Change the shell's working directory to directory *name*. If no argument is given then change to the home directory of the user.
If *name* is not found as a subdirectory of the current directory (and does not begin with '/', './' or '../'), then each component of the variable *cdpath* is checked to see if it has a subdirectory *name*. Finally, if all else fails but *name* is a shell variable whose value begins with '/', then this is tried to see if it is a directory.

continue

Continue execution of the nearest enclosing *while* or *foreach*. The rest of the commands on the current line are executed.

default:

Labels the default case in a *switch* statement. The default should come after all *case* labels.

dirs

Prints the directory stack; the top of the stack is at the left, the first directory in the stack being the current directory.

echo wordlist
echo **–n** wordlist

The specified words are written to the shells standard output, separated by spaces, and terminated with a newline unless the **–n** option is specified.

else
end
endif
endsw

See the description of the *foreach*, *if*, *switch*, and *while* statements below.

eval arg ...

(As in *sh*(1).) The arguments are read as input to the shell and the resulting command(s) executed in the context of the current shell. This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions.

exec command

The specified command is executed in place of the current shell.

exit
exit(expr)

The shell exits either with the value of the *status* variable (first form) or with the value of the specified *expr* (second form).

foreach name (wordlist)
...

end

The variable *name* is successively set to each member of *wordlist* and the sequence of commands between this command and the matching *end* are executed. (Both *foreach* and *end* must appear alone on separate lines.)

The builtin command *continue* may be used to continue the loop prematurely and the builtin command *break* to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with '?' before any statements in the loop are executed. If you make a mistake typing in a loop at the terminal you can rub it out.

glob wordlist

Like *echo* but no '\' escapes are recognized and words are delimited by null characters in the output. Useful for programs which wish to use the shell to filename expand a list of words.

goto word

The specified *word* is filename and command expanded to yield a string of the form 'label'. The shell rewinds its input as much as possible and searches for a line of the form 'label:' possibly preceded by blanks or tabs. Execution continues after the specified line.

hashstat

Print a statistics line indicating how effective the internal hash table has been at locating commands (and avoiding *exec*'s). NOTE: **hashstat** is not available in this release. An *exec* is attempted for each component of the *path* where the hash function indicates a possible hit, and in each component which does not begin with a '/'.

history
history *n*
history –r *n*
history –h *n*

Displays the history event list; if **n** is given only the *n* most recent events are printed. The **–r** option reverses the order of printout to be most recent first rather than oldest first. The **–h** option causes the history list to be printed without leading numbers. This is used to produce files suitable for sourceing using the –h option to *source*.

if (expr) command

If the specified expression evaluates true, then the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *Command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Input/output redirection occurs even if *expr* is false, when command is **not** executed (this is a bug).

if (expr) **then**
...
else if (expr2) then
...
else

...

endif

If the specified *expr* is true then the commands to the first *else* are executed; otherwise if *expr2* is true then the commands to the second **else** are executed, etc. Any number of *else-if* pairs are possible; only one *endif* is needed. The *else* part is likewise optional. (The words *else* and *endif* must appear at the beginning of input lines; the *if* must appear alone on its input line or after an *else*.)

jobs
jobs –l

Lists the active jobs; given the –l options lists process id's in addition to the normal information.

kill pid
kill –sig pid ...
kill –l

Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in */usr/include/signal.h*, stripped of the prefix "SIG"). The signal names are listed by "kill –l". There is no default, saying just 'kill' does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal as well.

login

Terminate a login shell, replacing it with an instance of **/bin/login.** This is one way to log off, included for compatibility with *sh*(1).

logout

Terminate a login shell. Especially useful if *ignoreeof* is set.

nice
nice +number
nice command
nice +number command

The first form sets the scheduling priority for this shell to 4. The second form sets the priority to the given number. The final two forms run command at priority 4 and *number* respectively. The greater the number, the less cpu the process will get. The super-user may specify negative priority by using 'nice –number ...'. Command is always executed in a sub-shell, and the restrictions placed on commands in simple *if* statements apply.

nohup
nohup command

The first form can be used in shell scripts to cause hangups to be ignored for the remainder of the script. The second form causes the specified command to be run with hangups ignored. All processes detached with '&' are effectively *nohup'ed*.

onintr

onintr –
onintr label

Control the action of the shell on interrupts. The first form restores the default action of the shell on interrupts which is to terminate shell scripts or to return to the terminal command input level. The second form 'onintr –' causes all interrupts to be ignored. The final form causes the shell to execute a 'goto label' when an interrupt is received or a child process terminates because it was interrupted.

In any case, if the shell is running detached and interrupts are being ignored, all forms of *onintr* have no meaning and interrupts continue to be ignored by the shell and all invoked commands.

popd
popd +n

Pops the directory stack, returning to the new top directory. With an argument '+*n*' discards the *n*th entry in the stack. The elements of the directory stack are numbered from 0 starting at the top.

pushd
pushd name
pushd +n

With no arguments, *pushd* exchanges the top two elements of the directory stack. Given a *name* argument, *pushd* changes to the new directory (ala *cd*) and pushes the old current working directory (as in *csw*) onto the directory stack. With a numeric argument, rotates the *n*th argument 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 the internal hash table of the contents of the directories in the *path* variable to be recomputed. This is needed if new commands are added to directories in the *path* while you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories.

repeat count command

The specified *command* which is subject to the same restrictions as the *command* in the one line *if* statement above, is executed *count* times. I/O redirections occur exactly once, even if *count* is 0.

set
set name
set name=word
set name[index]=word
set name=(wordlist)

The first form of the command shows the value of all shell variables. Variables which have other than a single word as value print as a parenthesized word list. The second form sets *name* to the null string. The third form sets *name* to the single *word*. The fourth form sets the *index'th* component of name to word; this component must already exist.

The final form sets *name* to the list of words in *wordlist*. In all cases the value is command and filename expanded.

These arguments may be repeated to set multiple values in a single set command. Note however, that variable expansion happens for all arguments before any setting occurs.

setenv
setenv name
setenv name value
The first form lists all current environment variables. The last form sets the value of environment variable *name* to be *value*, a single string. The second form sets *name* to an empty string. The most commonly used environment variable USER, TERM, and PATH are automatically imported to and exported from the *csh* variables *user*, *term*, and *path*; there is no need to use *setenv* for these.

shift
shift variable
The members of *argv* are shifted to the left, discarding *argv[1]*. It is an error for *argv* not to be set or to have less than one word as value. The second form performs the same function on the specified variable.

source name
source –h name
The shell reads commands from *name*. *Source* commands may be nested; if they are nested too deeply the shell may run out of file descriptors. An error in a *source* at any level terminates all nested *source* commands. Normally input during *source* commands is not placed on the history list; the –h option causes the commands to be placed in the history list without being executed.

switch (string)
case str1:
...
breaksw
...
default:
...
breaksw
endsw
Each case label is successively matched, against the specified *string* which is first command and filename expanded. The file metacharacters '*', '?' and '[...]' may be used in the case labels, which are variable expanded. If none of the labels match before a 'default' label is found, then the execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command *breaksw* causes execution to continue after the *endsw*. Otherwise control may fall through case labels and default labels as in C. If no label matches and there is no default, execution continues after the *endsw*.

umask
umask value
The file creation mask is displayed (first form) or set to the specified value (second form). The mask is given in octal. Common values for the mask are 002 giving all access to the group and read and execute access to others or 022 giving all access except no write access for users in the group or others.

unalias pattern
All aliases whose names match the specified pattern are discarded. Thus all aliases are removed by 'unalias *'. It is not an error for nothing to be *unaliased*.

unhash
Use of the internal hash table to speed location of executed programs is disabled.

unset pattern
All variables whose names match the specified pattern are removed. Thus all variables are removed by 'unset *'; this has noticeably distasteful side-effects. It is not an error for nothing to be *unset*.

unsetenv pattern
Removes all variables whose name match the specified pattern from the environment. See also the *setenv* command above and *printenv*(1).

wait
All background jobs are waited for. It the shell is interactive, then an interrupt can disrupt the wait, at which time the shell prints names and job numbers of all jobs known to be outstanding.

while (expr)
...
end
While the specified expression evaluates non-zero, the commands between the *while* and the matching end are evaluated. *Break* and *continue* may be used to terminate or continue the loop prematurely. (The *while* and *end* must appear alone on their input lines.) Prompting occurs here the first time through the loop as for the *foreach* statement if the input is a terminal.

@
@ name = expr
@ name[index] = expr
The first form prints the values of all the shell variables. The second form sets the specified *name* to the value of *expr*. If the expression contains '<', '>', '&' or '|' then at least this part of the expression must be placed within '(' ')'. The third form assigns the value of *expr* to the *index'th* argument of *name*. Both *name* and its *index'th* component must already exist.

The operators '*=', '+=', etc are available as in C. The space separating the name from the assignment operator is optional. Spaces are,

however, mandatory in separating components of *expr* which would otherwise be single words.

Special postfix '++' and '--' operators increment and decrement *name* respectively, i.e. '@ i++'.

Pre-defined and environment variables

The following variables have special meaning to the shell. Of these, *argv*, *cwd*, *home*, *path*, *prompt*, *shell* and *status* are always set by the shell. Except for *cwd* and *status* this setting occurs only at initialization; these variables will not then be modified unless this is done explicitly by the user.

This shell copies the environment variable USER into the variable *user*, TERM into *term*, and HOME into *home*, and copies these back into the environment whenever the normal shell variables are reset. The environment variable PATH is likewise handled; it is not necessary to worry about its setting other than in the file *.cshrc* as inferior *csh* processes will import the definition of *path* from the environment, and re-export it if you then change it.

argv Set to the arguments to the shell, it is from this variable that positional parameters are substituted, i.e. '$1' is replaced by '$argv[1]', etc.

cdpath Gives a list of alternate directories searched to find subdirectories in *chdir* commands.

cwd The full pathname of the current directory.

echo Set when the –x command line option is given. Causes each command and its arguments to be echoed just before it is executed. For non-builtin commands all expansions occur before echoing. Builtin commands are echoed before command and filename substitution, since these substitutions are then done selectively.

histchars Can be given a string value to change the characters used in history substitution. The first character of its value is used as the history substitution character, replacing the default character !. The second character of its value replaces the character ^ in quick substitutions.

history Can be given a numeric value to control the size of the history list. Any command which has been referenced in this many events will not be discarded. Too large values of *history* may run the shell out of memory. The last executed command is always saved on the history list.

home The home directory of the invoker, initialized from the environment. The filename expansion of '~' refers to this variable.

ignoreeof If set the shell ignores end-of-file from input devices which are terminals. This prevents shells from accidentally being killed by control-D's.

mail The files where the shell checks for mail. This is done after each command completion which will result in a prompt, if a specified interval has elapsed. The shell says 'You have new mail.' if the file exists with an access time not greater than its modify time.

If the first word of the value of *mail* is numeric it specifies a different mail checking interval, in seconds, than the default, which is 10 minutes.

If multiple mail files are specified, then the shell says 'New mail in *name*' when there is mail in the file *name*.

noclobber As described in the section on *Input/output*, restrictions are placed on output redirection to insure that files are not accidentally destroyed, and that '>>' redirections refer to existing files.

noglob If set, filename expansion is inhibited. This is most useful in shell scripts which are not dealing with filenames, or after a list of filenames has been obtained and further expansions are not desirable.

nonomatch If set, it is not an error for a filename expansion to not match any existing files; rather the primitive pattern is returned. It is still an error for the primitive pattern to be malformed, i.e. 'echo [' still gives an error.

notify If set, the shell notifies asynchronously of job completions. The default is to rather present job completions just before printing a prompt.

path Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no *path* variable then only full path names will execute. The usual search path is '.', '/bin' and '/usr/bin', but this may vary from system to system. For the super-user the default search path is '/etc', '/bin' and '/usr/bin'. A shell which is given neither the **–c** nor the **–t** option will normally hash the contents of the directories in the *path* variable after reading *.cshrc*, and each time the *path* variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the *rehash* or the commands may not be found.

prompt The string which is printed before each command is read from an interactive terminal input. If a '!' appears in the string it will be replaced by the current event number unless a preceding '\' is given. Default is '% ', or '# ' for the super-user.

savehist is given a numeric value to control the number of entries of the history list that are saved in ~/.history when the user logs out. Any command which has been referenced in this many

events will be saved. During start up the shell sources ~/.history into the history list enabling history to be saved across logins. Too large values of *savehist* will slow down the shell during start up.

shell The file in which the shell resides. This is used in forking shells to interpret files which have execute bits set, but which are not executable by the system. (See the description of *Non-builtin Command Execution* below.) Initialized to the (system-dependent) home of the shell.

status The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Builtin commands which fail return exit status '1', all other builtin commands set status '0'.

verbose Set by the –v command line option, causes the words of each command to be printed after history substitution.

Non-builtin command execution

When a command to be executed is found to not be a builtin command the shell attempts to execute the command via *execv*(2). Each word in the variable *path* names a directory from which the shell will attempt to execute the command. If it is given neither a –c nor a –t option, the shell will hash the names in these directories into an internal table so that it will only try an *exec* in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via *unhash*), or if the shell was given a –c or –t argument, and in any case for each directory component of *path* which does not begin with a '/', the shell concatenates with the given command name to form a path name of a file which it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus '(cd ; pwd) ; pwd' prints the *home* directory; leaving you where you were (printing this after the home directory), while 'cd ; pwd' leaves you in the *home* directory. Parenthesized commands are most often used to prevent *chdir* from affecting the current shell.

If the file has execute permissions but is not an executable binary to the system, then it is assumed to be a file containing shell commands and a new shell is spawned to read it.

If there is an *alias* for *shell* then the words of the alias will be prepended to the argument list to form the shell command. The first word of the *alias* should be the full path name of the shell (e.g. '$shell'). Note that this is a special, late occurring, case of *alias* substitution, and only allows words to be prepended to the argument list without modification.

Argument list processing

If argument 0 to the shell starts with a '–' then this is a login shell. The flag arguments are interpreted as follows:

–b This flag forces a "break" from option processing, causing any further shell arguments to be treated as non-option arguments. The remaining arguments will not be interpreted as shell options. This may be used to pass options to a shell script without confusion or possible subterfuge. The shell will not run a set-user ID script without this option.

–c Commands are read from the (single) following argument which must be present. Any remaining arguments are placed in *argv*.

–e The shell exits if any invoked command terminates abnormally or yields a non-zero exit status.

–f The shell will start faster, because it will neither search for nor execute commands from the file '.cshrc' in the invoker's home directory.

–i The shell is interactive and prompts for its top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their inputs and outputs are terminals.

–n Commands are parsed, but not executed. This aids in syntactic checking of shell scripts.

–s Command input is taken from the standard input.

–t A single line of input is read and executed. A '\' may be used to escape the newline at the end of this line and continue onto another line.

–v Causes the *verbose* variable to be set, with the effect that command input is echoed after history substitution.

–x Causes the *echo* variable to be set, so that commands are echoed immediately before execution.

–V Causes the *verbose* variable to be set even before '.cshrc' is executed.

–X Is to –x as –V is to –v.

After processing of flag arguments, if arguments remain but none of the **–c**, **–i**, **–s**, or **–t** options was given, the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by '$0'. Since many systems use either the standard version 6 or version 7 shells whose shell scripts are not compatible with this shell, the shell will execute such a 'standard' shell if the first character of a script is not a '#', i.e. if the script does not start with a comment. Remaining arguments initialize the variable *argv*.

Signal handling

The shell normally ignores *quit* signals. Jobs running detached (by '&') are immune to signals generated from the keyboard, including hangups. Other signals have the values which the shell inherited from its parent. The shells handling of interrupts and terminate signals in shell scripts can be controlled by *onintr*. Login shells catch the *terminate* signal; otherwise this signal is

passed on to children from the state in the shell's parent. In no case are interrupts allowed when a login shell is reading the file '.logout'.

FILES

~/.cshrc	Read at beginning of execution by each shell.
~/.history	Holds history. See tje *savehist* command.
~/.login	Read by login shell, after '.cshrc' at login.
~/.logout	Read by login shell, at logout.
/bin/sh	Standard shell, for shell scripts not starting with a '#
/tmp/sh*	Temporary file for '<<'.
/etc/passwd	Source of home directories for '~name'.

LIMITATIONS

In ARIX-OS V.3, words can be no longer than 1024 characters. The system limits argument lists to 5120 characters. The number of arguments to a command which involves filename expansion is limited to 1/6'th the number of characters allowed in an argument list. Command substitutions may substitute no more characters than are allowed in an argument list. To detect looping, the shell restricts the number of *alias* substitutions on a single line to 20.

SEE ALSO

sh(1), access(2), exec(2), fork(2), kill(2), pipe(2), sigset(2), umask(2), wait(2), tty(4), a.out(4), environ(5).

CAVEATS

The following items are not supported in this release of **csh**:

Job control facilities (**^Z, ^Y, suspend, fg, by, %job, %job&, stop**),

Timing/resource monitoring (**limit, unlimit, time**),

File name completion

alloc,

hashstat.

Alias substitution is most often used to clumsily simulate shell procedures; shell procedures should be provided rather than aliases.

Commands within loops, prompted for by '?', are not placed in the *history* list. Control structure should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with '|', and to be used with '&' and ';' metasyntax.

It should be possible to use the ':' modifiers on the output of command substitutions. All and more than one ':' modifier should be allowed on '$' substitutions. The builtin command **wait** does not allow an interrupt by "SIGINT" (the delete key).

TECHNICAL NOTES

1. Typically, the *make* command expects the Bourne shell to be used as the shell environment. If a "Variable Syntax" error message results from running **make**(1), the following suggestions are offered:

a. Change the environment using the command:

setenv SHELL /bin/sh

b. Specify the shell to be used in the makefile by adding the following line to **make**:

SHELL=/bin/sh

AND DO NOT USE THE **-e** option of **make**.

c. Specify the shell to be used on the command line:

make .... SHELL=/bin/sh

(Command line definitions override environment definitions, so the **-e** is not a problem.)

2. All "stty" commands must be placed in the .login file, not in the .cshrc file; otherwise, error messages will occur under certain circumstances.

3. To echo a carriage return in csh, use:

echo <backslash><return>

rather than

echo "<backslash>n"

NAME

system – format of ARIX computer system description file

DESCRIPTION

This file contains information about the hardware configuration and system-dependent parameters for the user's system. A more complete description of the system file is found in "Setting up the ARIX-OS" in the *System Administrator's Guide*. This information is used by the *config*(1M) program in configuring systems. The file is divided into two sections, separated by a line with a dollar sign (**$**) in column 1. The first section describes the hardware configuration and the second contains system-dependent information. Any lines with a number sign (**#**) in column 1 are treated as comments and are ignored. Blank lines are also ignored. All fields may be separated by one or more space and tab characters.

System-Dependent Information

This section specifies ARIX-OS devices, ARIX-OS parameters and software drivers.

The root and pipe devices are specified by:

root devname minor
pipe devname minor

The swap device is specified by:

swap devname minor low count

Tunable parameters are specified by:

parm value

Software drivers are specified in one of two forms:

driver num
driver

SEE ALSO

master(4).
config(1M), don(1M) in the *System Administrator's Reference Manual.*
Setting up the ARIX-OS in the *System Administrator's Guide.*

Support

Warranty and Support Agreement assistance for this software product is available from ARIX Customer Support. All calls for support should be made directly to our support department which may be contacted by telephone at:

- **800-237-2783** from outside California,
- **800-521-5783** from inside California, outside the 408 area code, or
- **408-432-1200** and ask for Customer Support.