CHAPTER 1

# About This Book

This guide provides detailed information on commands available from the command line interface (CLI), and a table that correlates the attributes used in the CLI, the configuration program, and the Management Information Base (MIB). It provides a description of the MIB used to manage LightStream 2020 enterprise ATM switches. It also contains manual pages for LynxOS commands and a user's guide for the vi text editor.

## Audience

The *LightStream 2020 Command and Attribute Reference Guide* is intended for anyone who operates a LightStream network.

Users of the LightStream document set are expected to have a general understanding of basic data communications concepts and some knowledge of UNIX.

## Organization

This guide is organized as follows:

- About This Book — Describes the audience, organization, and conventions for this book.
- CLI Command Reference — Lists all CLI commands and provides detailed descriptions of syntax, arguments, etc.
- Setting and Displaying Configuration Attributes — Lists the configuration parameters that you can configure with the LightStream configuration tool, and shows the functionally equivalent CLI commands for displaying and setting a node's configuration if the configuration tool is not available to you.
- LightStream MIB Reference — Shows the complete LightStream enterprise MIB and lists the address and type of each attribute.
- LynxOS Command Reference — Provides manual pages for LynxOS commands that you might need to operate your LightStream network.
- Vi Editor User's Guide — Provides complete user information for the vi text editor.
- BASH Shell Reference — This appendix provides instructions on using the bash shell.

## Related Documentation

The following is a list of LightStream manuals and other material relevant to LightStream users.

- *LightStream 2020 System Overview*

The system overview explains what a LightStream switch is and how it works. It outlines ATM technology and describes LightStream hardware and software.

- *LightStream 2020 Site Planning and Cabling Guide*

  The site planning and cabling guide (SPCG) tells you how to prepare your site to receive LightStream hardware. It includes space, environmental and electrical requirements, rack selection guidelines, requirements for the management workstation, and information on cables and connectors.

- *LightStream 2020 Installation and Troubleshooting Manual*

  The installation and troubleshooting manual (I&TM) tells you how to install LightStream hardware and software, how to diagnose hardware problems, and how to replace faulty hardware components.

- *LightStream 2020 Configuration Guide*

  The configuration guide provides the information you need to configure LightStream switches. It describes the configuration tools and how to use them. It describes the configuration database and defines all configurable attributes and their settings. The guide also provides step-by-step configuration procedures.

- *LightStream 2020 Operations Guide*

  The operations guide is a task-oriented book that tells you how to operate a network of LightStream switches. The guide presents an overview of network operations tasks, describes the command line interface (CLI), and presents procedures for performing monitor and control tasks such as displaying the status of nodes, cards and ports, viewing statistics, and creating collections of traffic data.

- *LightStream 2020 Administration Guide*

  The administration guide describes LightStream network management functions such as setting up a new network, troubleshooting, and optimizing the load across trunks. The guide describes network management tools, then presents step-by-step procedures for performing the functions.

- *LightStream 2020 Traps Reference Manual*

  This manual presents an overview of LightStream traps (error and event messages) and a list of operational, SNMP, and informational traps generated by the LightStream switch.

- *LightStream 2020 Command Line Interface (CLI) Reference Card*

  The reference card compactly summarizes the syntax and arguments of all CLI commands.

- *LightStream 2020 Release Notes*

  The release notes provide a software upgrade procedure and describe new features and special considerations, including information on known software bugs.

**Note** The release notes contain important information that does not appear in other documents.

- *LightStream 2020 Online Help*

  The LightStream command line interface (CLI) and configuration program both produce online help facilities.

Before attempting to install, configure, operate, or troubleshoot a network of LightStream switches, read the *LightStream 2020 System Overview*. This overview provides important background information about the LightStream product and the ATM technology on which the product is based. Read the *LightStream 2020 System Overview* first. Then use Table 1-1 to determine which manuals you should read next.

**Table 1-1 LightStream Reading Path**

| If you want to: | Read the following manuals in the order listed below: |
|---|---|
| Install LightStream switches | *LightStream 2020 Release Notes*[1] |
| | *LightStream 2020 Site Planning and Cabling Guide* |
| | *LightStream 2020 Installation and Troubleshooting Manual* |
| Configure LightStream switches | *LightStream 2020 Release Notes*[1] |
| | *LightStream 2020 Configuration Guide* |
| | *LightStream 2020 Online Help Screens* |
| Set up or expand a LightStream network | *LightStream 2020 Release Notes*[1] |
| | *LightStream 2020 Administration Guide* |
| | *LightStream 2020 Online Help Screens* |
| Operate a LightStream network | *LightStream 2020 Release Notes*[1] |
| | *LightStream 2020 Operations Guide* |
| | *LightStream 2020 Command and Attribute Reference Guide* |
| | *LightStream 2020 Command Line Interface (CLI) Reference Card* |
| | *LightStream 2020 Traps Reference Manual* |
| | *LightStream 2020 Online Help Screens* |
| Manage or troubleshoot a LightStream network | *LightStream 2020 Release Notes*[1] |
| | *LightStream 2020 Operations Guide* |
| | *LightStream 2020 Administration Guide* |
| | *LightStream 2020 Command and Attribute Reference Guide* |
| | *LightStream 2020 Command Line Interface (CLI) Reference Card* |
| | *LightStream 2020 Traps Reference Manual* |
| | *LightStream 2020 Online Help Screens* |
| Troubleshoot LightStream hardware | *LightStream 2020 Release Notes*[1] |
| | *LightStream 2020 Installation and Troubleshooting Manual* |
| | *LightStream 2020 Site Planning and Cabling Guide* |

1. We recommend that you review the release notes before attempting to install, configure, operate, or troubleshoot a LightStream switch. The release notes contain important information that does not appear in other documents.

# Text Conventions

Table 1-2 describes conventions used to distinguish different types of text:

**Table 1-2 Text Conventions**

| Convention | Purpose | Example |
|---|---|---|
| **`Bold screen literal`** type | Represents user input. | `$` **`date`** |
| `Screen literal` type | Represents system output | `Wed May 6 17:01:03 EDT 1994` |
| **Boldface** type | Denotes names of commands, command arguments, and switches. Command names are case sensitive; enter them exactly as they appear in the text. | Issue the **clear** command. |
| *Italic* type | Used for titles of documents and for emphasis. | *LightStream 2020 Configuration Guide*<br>File names are *case* sensitive. |
| Angle brackets < > | Indicate user-specified parameters or classes of user responses. When you see this notation in a syntax statement, make the substitution but do not type the angle brackets. | If you see:<br>**set port** <c.p> <state><br>you might type:<br>**set port 4.3 active** |
| Square brackets [ ] | Indicate keys on the keyboard, or optional arguments or parameters for commands. You can omit optional arguments and parameters in any command. | Press **[Return]**.<br>`cli>` **help** [<topic>] |
| Caret symbol ^ | When the caret symbol precedes a character, it refers to the control key. | **^X** is the same as **[Control] X** |
| Curly braces { } | Indicate a choice of arguments or parameters for commands. Arguments or parameters are separated by a vertical line {\|}, and you must select *one*. | `cli>` **set cli traplevel {off\|info\|oper\|trace\|debug}** |

**2 x 2 x**

# CLI Command Reference

The LightStream 2020 enterprise ATM switch supports a command line interface (CLI). This chapter lists all the CLI commands in alphabetical order, and provides detailed information on each command.

The CLI commands may be grouped by function as shown in Table 2-1:

**Table 2-1 CLI Commands and Functions**

| Command Type | Function |
|---|---|
| CLI control commands | Monitoring and control |
| MIB commands | Advanced monitoring and control |
| VLI commands | Virtual LAN internetworking |
| Diagnostics commands | Diagnostic tests |

If you can identify in Table 2-1 the type of action that you wish to perform, refer to Table 2-2 for the commands that you can use to perform that action. Table 2-2 describes the various CLI commands. The commands that you are most likely to use for normal day-to-day operation are the CLI control commands and the MIB commands.

In Table 2-2, the monitoring and control commands **set** and **show** are singled out and separated from the CLI control commands because of their complexity.

**Table 2-2 CLI Commands**

| Type | Name | Function |
|---|---|---|
| The set Command | **set** | Change the state of the specified attribute. See the section “The set Command.” |
| The show Command | **show** | Display the value of the specified attribute(s). See the section “The show Command.” |
| CLI Control Commands | **clear** | Clear the screen. |
| | **exit** | Exit CLI or protected mode. |
| | **help** | Display CLI help information. |
| | **password** | Change the password for protected mode. |
| | **ping** | Send ICMP echo packets to a host and report on any returned packets. |
| | **protected** | Enter protected mode. |
| | **quit** | Exit CLI or protected mode. |
| | **shell** | Execute a LynxOS command under a copy of the LynxOS shell. |
| | **source** | Execute a CLI script (CLI commands stored in a disk file). |

| Type | Name | Function |
|---|---|---|
| MIB Commands | **browse** | Browse the MIB tree. |
| | **getsnmp** | Display the value of a MIB object. |
| | **getnextsnmp** | Display the value of the object in the MIB tree that follows the specified object. |
| | **setsnmp** | Change the state of the specified MIB object. |
| | **walksnmp** | Display the values of all MIB objects in the MIB tree starting with the specified object. |
| VLI Commands | **define** | Define a bridge filter. |
| | **delete** | Delete a bridge filter. |
| Diagnostics Commands | **connect** | Logically attach the console or modem I/O ports to a given card within a LightStream node. |
| | **loadcard** | Load the specified file into the specified card, start the card, and establish a console connection between the CLI and the TCS slave on the card. |
| | **test** | Run field diagnostics tests from the CLI. |

Two other commands are described in the "Diagnostics Commands" section. They require detailed knowledge of the contents and functions of hardware registers and memory locations. These are the commands **read** and **write**.

The CLI help facility lists all the commands alphabetically. Refer to Chapter 3 of the *LightStream 2020 Operations Guide* for a detailed description of the online help facility in the CLI.

# The set Command

Use the **set** command to set the value of a specified MIB object within a LightStream node, or to set the state of the CLI program.

## Syntax

**set** *type* **[***ID***]** *parameter1* **[***parameter2***]**

## Arguments

The *type* argument may be any of those shown in Table 2-3:

**Table 2-3 type Arguments**

| ***type* Argument** | **Function** |
|---|---|
| **`set card`** | Per-card attributes |
| **`set chassis`** | Chassis-wide attributes |
| **`set cli`** | CLI attributes |
| **`set collection`** | Collection records |
| **`set config`** | Configuration changes saved in local database |
| **`set modem`** | Modem attributes |

| *type* Argument | Function |
|---|---|
| `set pid` | Per-process attributes |
| `set port` | Per-port attributes |
| `set snmp` | SNMP control attributes |
| `set stb` | Spanning-tree bridge attributes |
| `set tcs` | Per-card attributes under TCS |
| `set trap` | Display of traps |

**Note** The **set** command requires protected mode for **set modem**, **set tcs** and **set trap** only.

**Note** The **set** command requires that the read/write community name be set first to a name that has been assigned the value write in the mma.communities file (unless *parameter1* is cli, modem, or snmp). Because the default community name "public" is read only, the **set** command fails if the read/write community name has not been set first. See the description of the command **set snmp community** and Chapter 6 of the *LightStream 2020 Operations Guide* for information on setting the read/write community.

The additional arguments that may be used with each *type* argument are explained below.

# set card

Set the administrative state of the card to active, inactive, or testing.

## Syntax

**set card** *card#* **{active|inactive|testing}**

## Arguments

- *card#*

  A card number.

- **active**

  Set the administrative state of the specified card to active.

**Note** When the card is set active from some other state, card parameters are set to defaults, then overwritten from on-board memory (if temporary changes were made) and from the configuration database, in that order. The result can be a combination of defaults, "temporary" changes, and database settings, depending upon which parameters were set in EEPROM and in the configuration database.

**Note** After a power reset or reboot of the node, the operational status of a card may be down while its administrative status and configuration register value are both up. To bring the card up in these circumstances, set it to **inactive** and then to **active**.

- **inactive**

  Set the administrative state of the specified card to **inactive**.

**Note** Do not use the Verify function of the configurator when a card is set to **inactive**. The Verify function copies attribute values from run-time memory. When a card is inactive (or down for any reason), the Verify function can access only the card's type, number, and administrative status. If you choose to write values to the local database, it deletes all other configured attribute values stored there. See the *LightStream 2020 Configuration Guide* for details about the Verify function.

- **testing**

  Set the administrative state of the specified card to **testing**. This is done during some troubleshooting procedures. The **test** command sets the card state to testing.

# set chassis

Set values of specified chassis attributes.

## Syntax

**set chassis** *attribute* **[***value* **[***interval ms***]]**

## Arguments

- **activeip** *IPaddress*

  **secondaryip** *IPaddress*

  Set the IP addresses that are used for network management. The active IP address designates whichever NP is active in a given chassis; the secondary IP address designates the backup NP. These addresses are known to both NPs and to all nodes in the LightStream network. To connect to either address via a host or router outside the LightStream network, the address must be included in the static routing table on that host or router.

- **congestion** *interval ms*

  Set three time values used to control congestion avoidance operations. The *ms* argument is a number of microseconds, and the *interval* arguments are as follows:

  — **maxpermitinterval** *ms*

  **minpermitinterval** *ms*

  The maximum (minimum) interval, in microseconds, at which trunk cards and outgoing edge cards may report permit limits.

  — **mincainfointerval** *ms*

The minimum interval, in microseconds, at which congestion avoidance processes may distribute aggregated CA updates to input edge cards.

- **consoletraplevel {off|oper|info|trace|debug}**

  Set the level of traps that are reported by this node to the console. The info level includes oper traps, trace includes info and oper traps, and debug includes all traps. See the *LightStream 2020 Traps Reference Manual* for information about trap levels.

---

**Note** There must be a compelling reason to use any arguments other than **off** or **oper**. Use **set trap** for individual traps instead, to avoid flooding the node with traps, which could cripple it.

---

- **defrouter** *IPaddress*

  Set the default router address for network management traffic originating at the local NP. This address is used in the absence of any other routing information for such traffic.

- **ethernetaddr** *etheraddr*

  Set the Ethernet address for the NP. It is used by whichever NP is active. Not all LightStream nodes need have an Ethernet connection.

- **ethernetmask** *mask*

  Set the subnet mask for the Ethernet address.

- **name** *chassis_name*

  Set the chassis name (node name).

- **netmask** *mask*

  Set the subnet mask for the active and secondary IP addresses.

- **primaryswitch {switcha|switchb}**

  Determine which switch, SA or SB, is the primary switch.

---

**Note** This operation resets all cards on the affected device.

---

- **traplevel {oper|info|trace|debug}**

  Set the level of traps that are reported for this node. The info level includes oper traps, trace includes info and oper traps, and debug includes all traps. See the *LightStream 2020 Traps Reference Manual* for information about trap levels.

---

**Note** There must be a compelling reason to use any arguments other than **off** or **oper**. Use **set trap** for individual traps instead to avoid flooding the node with traps, which could cripple it.

---

- **traplog {on|off}**

  Turn the logging of traps in the traplog file on or off.

# set cli

Set values of specified CLI attributes.

## Syntax

**set cli** *attribute* **[***value* **[***interval ms***]]**

## Arguments

- **debug {on|off}**

  If the debug flag is on, additional information about the course of command execution is displayed, including the names of MIB variables as they are queried or set, and each trap message becomes quite verbose.

- **echosource {on|off}**

  Turn the echoing of sourced commands on or off. The default is to display shell commands as they are executed under the **source** command.

- **lineedit {on|off}**

  Turn line editing capability with control keys on or off.

- **log {"***logfile***"|off}**

  Turn the CLI logging function on by directing its output to the specified file *logfile*. The file *logfile* must be in the current directory (usually the same directory as the user account you are using). If it is not, you must enter the full pathname of the file. All user input and output of the current CLI session is copied to *logfile* until you turn the logging function off with **set cli log off** or exit the CLI. (The new output cannot be displayed at the bash prompt from another window until this happens.) If you re-open the same log file, the new session is appended to the existing file.

---

**Note** Always surround the file name or pathname of *logfile* with quotation marks, as in the following example:

---

cli> **set cli log "cli.log.9502"**

- **term** *termtype*

  Set the terminal type to *termtype*. See the file /etc/termcap for possible *termtype* values.

- **timer**

  Reinitialize the timer that normally indicates time elapsed since the current CLI session was started.

- **traplevel {off|oper|info|trace|debug}**

  Set the level of traps that are reported to the CLI and to an NMS. The info level includes oper traps, trace includes info and oper traps, and debug includes all traps. See the *LightStream 2020 Traps Reference Manual* for information about trap levels.

**Note** There must be a compelling reason to use any arguments other than **off** or **oper**. Use **set trap** for individual traps instead, to avoid flooding the node with traps, which could cripple it.

# set collection

Create, configure, or control the specified collection process.

## Syntax

**set collection** *collection#* **{***action* **[***MIB_object***] |** *attribute value***}**

See Chapter 6 of the *LightStream 2020 Operations Guide* for information about data collections and how to use them, and see Chapter 5 of the *Operations Guide* for information about monitoring collections.

## Action Arguments of set collection

- **addvar** *MIB_object*

  Add *MIB_object* to the objects subject to the specified collection.

- **create**

  Create the specified collection.

- **del**

  Remove the specified collection from the system.

- **delvar** *MIB_object*

  Remove *MIB_object* from the set of objects subject to the specified collection.

- **halt**

  Halt the specified collection.

- **start**

  Start the specified collection.

## Attribute Arguments of set collection

- **begintime [[[***yy***:]***mm***:]***dd***:]***hh***:***mm***:***ss***

  Set the time at which the specified collection begins. The default beginning time is the current time.

- **endtime [[[***yy***:]***mm***:]***dd***:]***hh***:***mm***:***ss***

  Set the time at which the specified collection ends. The default end time is December 31, 2037 23:59:59.

- **filesize** *bytes* **[:** *begintime* **[:** *endtime***]]**

  Set the maximum size of the collection file in kilobytes, the time at which the specified collection begins, and the time at which the specified collection ends. The optional times *begintime* and *endtime* are in [[[*yy*:]*mm*:]*dd*:]*hh*:*mm*:*ss* format. The default file size is 100 Kb. The collection file is a circular file: when the collection data attains the configured file size limit, the process begins overwriting the data in the file from the beginning.

- **frequency** *ss* **[:** *begintime* **[:** *endtime***]]**

  Set the frequency (in seconds) at which collection is to be done, the time at which the specified collection begins, and the time at which the specified collection ends. The optional times *begintime* and *endtime* are in [[[*yy*:]*mm*:]*dd*:]*hh*:*mm*:*ss* format. The default frequency is 60 seconds.

# set config

Control write access to the MMA configuration database.

## Syntax

**set config { lock | unlock }**

## Arguments

- **lock**

  All changes to configuration parameters are written to the disk, and other concurrent users are prevented from making configuration changes with CLI commands. The CLI issues a periodic reminder that the chassis is locked. The lock times out automatically two minutes after the termination of the CLI session in which the lock was issued.

- **unlock**

  Multiple users can concurrently make configuration changes with CLI commands, none of which are written to disk. This is the default.

## Description

Write configuration changes to the MMA database, and prevent other users from making configuration changes; or restore the default, so that the CLI affects configuration parameters in run-time memory only.

These commands are equivalent to **setsnmp mmaSetLock 3** (chassis locked) and **setsnmp mmaSetLock 1** (chassis unlocked). The command **setsnmp mmaSetLock 2** locks the chassis to other users, but does not write changes to disk. This is useful for making experimental changes without interference. When **setsnmp** is used to set the mmaSetLock object to 2 or 3, the lock automatically times out after two minutes of no input from the user. With these commands, in contrast with the **set config lock** command, the CLI does not issue a periodic reminder that the chassis is locked.

If other users of the CLI attempt to use CLI set commands while the MMA is locked, they see the following generic SNMP error message:

```
SNMP error
```

**Note** After you make configuration changes and write them to the disk, as described above, the local database is out of synch with the global database. As soon as possible, use the verify function in the configuration tool on the network management station to copy configuration changes from the local configuration database on the LightStream node to the global configuration database on the network management station. The verify function retrieves the local settings and allows you to write them over the global values.

# set modem

Set the modem initialization string and modem password for the specified switch card.

## Syntax

**set modem {sa|sb} {initstring** *init_string*|**password** *password***}**

## Description

This command affects *only* the node on which the CLI is running when you execute it, regardless of a target set with the command **set snmp hostname** *name*.

---

**Note** The **set modem** command requires CLI protected mode. (See the **protected** command.)

---

# set pid

Set the trap level or administrative status of a process.

## Syntax

**set pid** *pid#* **{traplevel** *level* **|adminstatus {active|inactive} }**

## Arguments

- **traplevel {oper|info|trace|debug}**

  Set the level of traps that are reported for process number *pid#*. The info level includes oper traps, trace includes info and oper traps, and debug includes all traps. See the *LightStream 2020 Administration Guide* for information about the relationships between traps, PIDs, and processes. See the *LightStream 2020 Traps Reference Manual* for trap levels.

---

**Note** There must be a compelling reason to use any arguments other than **off** or **oper**. Use **set trap** for individual traps instead, to avoid flooding the node with traps, which could cripple it.

---

- **adminstatus {active|inactive}**

  Set the administrative status of process number *pid#* to active or inactive. When the operational status changes, the system restores it to this preferred state as soon as it can.

# set port

Configure a port.

## Description

Use the **set port** command to configure various attributes of a port. Most attributes can be configured only for an appropriate card type. The types of attributes include the port state (for all card types), bridging and VLI attributes, and protocol-specific characteristics. These are described under the following headings:

- State Arguments of set port
- Characteristics Arguments of set port
- ATM VCI Arguments of set port
- Bridge Arguments of set port
- Frame Forwarding Arguments of set port
- Frame Relay Arguments of set port
- FDDI Arguments of set port
- VLI Arguments of set port

## Syntax

**set port** *c.p arguments*

## State Arguments of set port

- **active|inactive|testing**

  Set the administrative state of the specified port to active, inactive, or testing. The **test** command sets the port state to testing.

- **loop {external|internal|remote}**

  Loop the specified port externally, internally, or remotely.

- **unloop**

  Unloop the specified port.

## Characteristics Arguments of set port

Configure or modify port characteristics. The arguments with the **characteristics** parameter are as follows:

- **characteristics csu {none|larse}**

  Set the CSU type to **larse** or specify that no CSU is present.

- **characteristics {dce-bitrate** *Kbits*|**dte-bitrate** *bits***}**

  Set the DCE or DTE bit rate for the specified port, depending upon the **dce-dte-type** value described below. The value of *Kbits* for the DCE bit rate may be 56, 64, 128, 192, 256, 384, 448, 512, 768, 896, 1344, 1536, 1792, 2688, 3584, 4000, or 5376. The value of *bits* for the DTE bit rate is unrestricted in the range of decimal integers 9,000 — 6,000,000.

- **characteristics dce-dte-type {dce|dte|dce-internal}**

Set the specified port to be a DCE, DTE, or DCE internal. The **dce** setting connects the receive clock to the TT interface signal. The **dce-internal** setting connects the receive clock to a locally generated clock. A DCE internal port is able to interface with DTE devices that cannot return the TT signal. This value is interdependent with the **dce-bitrate** or **dte-bitrate** value described above.

- **characteristics executechange**

  Make previously set administrative values operational for the specified port. The other arguments with the **characteristics** parameter set the administrative value only.

- **characteristics protocol {trunk|framerelay|frameforward|atm-uni}**

  Set the specified port to use the trunk protocol or one of the edge protocols (frame relay, frame forwarding, ATM-UNI).

---

**Note** Trunk and edge protocols cannot be intermixed on a single card.

---

**Note** Use the **set config lock** command before changing between **trunk** and any edge protocol. The reason is that the card resets and the value is read back from the local configuration database.

---

## ATM VCI Arguments of set port

- **set port** *c.p* **vci** *vci#* **{activate|deactivate|del|***attribute value***}**

  Activate or deactivate the ATM VCI on the specified port, or delete it. The VCI number must be in the range 1-32399 inclusive, and may be further restricted depending upon the type of line card. The VCI must be activated after setting VCI parameters. For the restrictions on the sequences in which these commands may be applied, refer to the *LightStream 2020 Administration Guide*. The ATM VCI arguments are as follows:

  — **vci** *vci#* **activate**

  Enable the specified VCI on the specified port after setting its parameters.

  — **vci** *vci#* **deactivate**

  Deactivate the specified VCI without deleting it, for example, keeping it as a backup circuit.

  — **vci** *vci#* **del**

  Deactivate and delete VCI *vci#* from the specified port.

  — **vci** *vci#* **destnode {***chassisID|IPaddress|chassisname***}**

  Set the destination node for ATM VCI on the specified port to a node identified by its chassis number, its IP address, or its chassis name (if previously set with **set chassis name**).

  — **vci** *vci#* **destport** *c.p*

  Set the destination port to *c.p* for the specified VCI.

  — **vci** *vci#* **destvci** *destvci#*

  Set the destination VCI to *destvci#* for the specified VCI. The VCI numbers *vci#* and *destvci#* must both be in the range 1-32399 inclusive.

  — **vci** *vci#* **insured-rate** *cells/sec*

Set the insured rate to *cells/sec* for the specified VCI.

— **vci** *vci#* **insured-burst** *cells*

Set the insured burst rate to *cells* for the specified VCI. The default is 128 cells.

— **vci** *vci#* **max-rate** *cells/sec*

Set the maximum rate to *cells/sec* for the specified VCI. The default rates are 109 cells/sec for MSC, 218 Cells/sec for CLC, and the line rate for LSC (frame forwarding and frame relay).

— **vci** *vci#* **max-burst** *cells*

Set the maximum burst rate to *cells* for the specified VCI. The default is 128 cells.

— **vci** *vci#* **bwtype {guaranteed|insured}**

Set the bandwidth type (cell-drop priority) on the primary portion of the specified VCI to guaranteed or insured. The default is insured.

— **vci** *vci#* **pri {0|1}**

Set the transmit priority of the specified VCI. This priority is used at each LightStream node in the VCI across the network. The default is 0 for frame relay circuits and for PVCs, and the default is 1 for frame forwarding circuits.

## Bridge Arguments of set port

The **set port** command has four bridge arguments: **bcast-limit**, **bflt**, **bflt-def**, and **stb**. The arguments are described below.

- **set port** *c.p* **bcast-limit {discard-all|forward-all|***packets/sec***}**

  Set the rate at which broadcast packets can be forwarded through this LAN port. Excess broadcast packets are dropped. To restore the default broadcast limit, enter this command with **-1** as the number of packets per second.

  The arguments used with the **bcast-limit** parameter are as follows:

  — **discard-all**

  Discard all broadcast packets sent to this port.

  — **forward-all**

  Forward all broadcast packets sent to this port.

  — *packets/sec*

  The maximum number of broadcast packets per second to be forwarded through this port, in the range 1-127.

- **set port** *c.p* **bflt** *ID* **{block** *priority*|**forward** *priority*|**del}**

  Associate a bridge filter with the specified port. The filter must already have been created with the **define** command. Up to 32 filters can be assigned to the same port (with a maximum of 1024 filters over all ports on a LightStream node), and a given filter can be associated with different ports.

---

**Note** These attributes are not affected when the card is reset.

---

The arguments used with the **bflt** parameter are as follows:

— *ID*

Identifies the filter that is to be associated with this port. The filter *ID* was assigned to the filter with the **define** command.

**{ block | forward }** *priority*

When a frame matches the filter, either block it or forward it, as indicated. The *priority* argument is a number that determines the sequence in which multiple filters are considered on this port. Each filter on a given port must have a unique priority number. The lowest number is considered first. We recommend assigning priority numbers by 100s (100, 200, 300, …), leaving large gaps for possible future insertions into the sequence.

**del**

Break the association between the specified bridge filter and the specified port. This must be done before the filter itself can be deleted with the **delete** command.

- **set port** *c.p* **bflt-def {block|forward}**

  Set the default bridging action for the specified port to **block** or **forward**.

---

**Note** This attribute is not affected when the card is reset.

---

- **set port** *c.p* **stb {pri** *#***|enable|disable|pathcost** *#***}**

  Set spanning-tree bridge parameters for the specified port (see also **set stb** *parameter value*).

---

**Note** These attributes are not affected when the card is reset.

---

The arguments are as follows:

— **pri** *#*

Set the priority of the specified port for a path using STP. The range is 0-255, and the default is 128.

— **{ enable | disable }**

Enable or disable bridge forwarding on the specified port. Ports are enabled when they come up, but the spanning tree protocol may disable ports to prevent loops if the topology of the bridged networks connected to this port changes.

— **pathcost** *#*

Set the cost of a path for the specified port (the contribution of this port to the path cost of those paths toward the root bridge that include it). The range is 1-65535, and the default value is calculated as 1000 divided by the speed of the network connection in Mbits/sec. Thus, Ethernet has a default cost of 100, FDDI has a default cost of 10.

## Frame Forwarding Arguments of set port

- **set port** *c.p* **frameforwarding {activate|deactivate|***attribute value***}**

Set the following frame forwarding parameters:

— **frameforwarding {activate | deactivate}**

Enable or disable the frame-forwarding circuit on the specified port.

— **frameforwarding destnode {***chassisID***|***chassisname***}**

Set the destination node for frame forwarding on the specified port to a node identified by its chassis ID or its chassis name (if previously set with **set chassis name**).

— **frameforwarding destport** *c.p*

Set the destination port to *c.p* for the specified port.

— **frameforwarding insured-rate** *bps*

Set the insured rate for the specified port to the bit rate *bps*.

— **frameforwarding insured-burst** *bytes*

Set the insured burst rate for the specified port to *bytes*.

— **frameforwarding max-rate** *bps*

Set the maximum rate for the specified port to the bit rate *bps*.

— **frameforwarding max-burst** *bytes*

Set the maximum burst rate for the specified port to *bytes*.

## Frame Relay Arguments of set port

The **set port** command has two frame relay arguments: **framerelay** and **dlci**. The arguments are described below.

- **set port** *c.p* **framerelay {lmiconfig|netinterfacetype}** *value*

  Set the frame relay parameters as follows:

  — **framerelay lmiconfig {none|frif|ansi_t1_617d|q933a}**

  Set the LMI configuration type to FRIF, ANSI T1 617D, or Q933A, or specify that there is no LMI for the specified port.

  — **framerelay netinterfacetype {uni|nni}**

  Set the frame relay net interface type to UNI or NNI for the specified port. Frame relay NNI is not supported in Release 2.0.

- **set port** *c.p* **dlci** *dlci#* **{***attribute value***|activate|deactivate|del}**

  Enable the circuit on the specified DLCI (between 16 and 991 inclusive), deactivate the specified DLCI, or remove the specified DLCI from the system. The arguments are as follows:

  — **dlci** *dlci#* **activate**

  Enable the circuit on the specified DLCI.

  — **dlci** *dlci#* **deactivate**

  Deactivate the specified DLCI.

  — **dlci** *dlci#* **del**

  Remove the specified DLCI from the system.

  — **dlci** *dlci#* **destnode {***chassisID***|***IPaddress***|***chassisname***}**

Set the destination node for the specified DLCI to a node identified by its chassis number, its IP address, or its chassis name (if previously set with **set chassis name**).

— **dlci** *dlci#* **destport** *c.p*

Set the destination port for the specified DLCI to port *c.p*.

— **dlci** *dlci#* **destdlci** *dlci#*

Set the destination DLCI for the specified DLCI to *dlci#* (a number between 16 and 991 inclusive).

— **dlci** *dlci#* **insured-rate** *bps*

Set the insured rate for the specified DLCI to the bit rate *bps*.

— **dlci** *dlci#* **insured-burst** *bytes*

Set the insured burst rate for the specified DLCI to *bytes*.

— **dlci** *dlci#* **max-rate** *bps*

Set the maximum rate for the specified DLCI to the bit rate *bps*.

— **dlci** *dlci#* **max-burst** *bytes*

Set the maximum burst rate for the specified DLCI to *bytes*.

## FDDI Arguments of set port

- **set port** *c.p* **fddi {{aport|bport}***parameter***|smt** *action***}**

Set FDDI port and station management (SMT) parameters as follows:

— **fddi {aport|bport}** *parameter*

Set the characteristics of FDDI port A or port B. The possible *parameter* values are as follows:

**action {enable|disable|start|stop}**

Enable, disable, start, or stop the specified FDDI port.

**connectpolicy {none|lct|loop|both}**

Specify the FDDI connection policy for this port. Use the **lct** argument for a MAC link competence test with the remote station (remote loop). Use the **loop** argument for an internal loop at the MAC. Enter **none** for neither internal nor external loop, or enter **both** for both.

**lercutoff** *error-rate*

Set the link error rate estimate at which an FDDI link connection on this port is broken. The range is 4-15 and the default is 7, meaning $10^{-7}$ errors per second.

---

**Note** In the unlikely event that this rate needs adjustment, it should be changed only by someone very knowledgeable about FDDI.

---

— **fddi smt** *action*

Set the characteristics of FDDI port A or port B, or set FDDI station management (SMT) parameters, as follows:

**t-notify** *sec*

Set the timer used in the Neighbor Notification protocol. The range is 2-30 seconds, and the default is 30 seconds.

---

**Note** In the unlikely event that this timer needs adjustment, it should be changed only by someone very knowledgeable about FDDI.

---

**stat-report {yes|no}**

If this switch is set to **yes** (the default), then Status Reporting Frames for FDDI events and conditions are sent to the SMT management software. Depending on your network management system, SMT may pass some of these on to higher levels, where they become visible to the operator as SNMP traps.

**station** *action*

Control the port as an FDDI station. The *action* parameter of a **set port** *c.p* **fddi smt station** *action* command may be any of the following:

**{connect|disconnect}**

Begin an FDDI connection sequence, or break an FDDI connection, respectively.

**path-test**

Test the viability of the FLDSUP path. This is not supported in the current release.

**{disable-a|disable-b}**

Disable the FDDI circuit on the A port or B port, respectively, if the other end of the link is not master (i.e. if the port mode is peer).

## VLI Arguments of set port

Set the Virtual LAN Internetworking (VLI) parameters. Only workgroups are currently supported.

- **set port** *c.p* **wgrp {add** *ID***[,** *ID***] | del {***ID***|all} | exclude | include }**

  Use these commands to maintain the workgroup list for each port. There is one list per port. The port is either *included in* or *excluded from* the listed workgroups (see **include** and **exclude**, below). The default is an include list that contains just workgroup 1. An empty include list is treated the same as this default case. An empty exclude list permits communication with every workgroup.

---

**Note** Run-time changes to the workgroup list are not affected when the card is reset.

---

— **add** *ID* **[,** *ID* . . .]

Add one or more workgroup IDs to the list for the specified port. There may be up to seven workgroup IDs per port (up to six in a list in the exclude sense). Use the **show port** *c.p* **wgrp** command to display the current list.

**Note** Adding a workgroup to the default list with this command does not delete the default workgroup automatically, unlike the configuration tool.

— **{ include | exclude }**

Change the sense of the workgroup list for the specified port, so that intercommunication is allowed with the workgroups in the list (**include**) or with all workgroups in the range 1—65,535 *except* those listed (**exclude**). An include list may name up to 7 workgroups per port, and an exclude list may name up to 6 workgroups per port. In practically all cases the workgroup list for a port should be defined in the include sense. The most important use of an exclude list is to enable a port to communicate with all workgroups. Do this by configuring an empty list (delete all workgroup IDs, including workgroup 1) and changing the sense to exclude.

**Note** Maintain a list of group IDs and the corresponding mnemonic workgroup names maintained by the configuration tool. Only the numeric IDs work with the **set** command.

— **del {** *ID* **| all }**

Delete the specified workgroup, or all workgroups, from the list for the specified port. *ID* is a workgroup ID number in the range 1-65,535. An empty include list is equivalent to the default group 1. An empty exclude list permits communication with every workgroup.

# set snmp

Set the value of SNMP attributes for the specified port.

## Syntax

**set snmp** *attribute value*

## Arguments

- **community** *name*

  Set the read-write community name to *name*. The **set** command requires that the read/write community name be set first to a name that is assigned the value write in the mma.communities file.

- **hostname {***name***|***IPaddress***}**

  Set the target host name to *name* or *IPaddress*. When the target is the node on which the CLI is running, *name* is **localhost**.

## Description

**Note** This command affects *only* the node on which the CLI is running when you execute it, regardless of a target set with the command **set snmp hostname** *name*.

# set stb

Define the spanning-tree bridge parameters for the node.

## Description

See also the command **set port** *c.p* **stb** *parameters*.

**Note** These attributes are not affected when the card is reset.

## Syntax

**set stb** *parameter value*

## Arguments

For the three arguments **maxage**, **forwdelay**, and **hellotimer**, the value is in hundredths of a second, truncated to seconds. For example, 400, 401, and 499 all signify 4 seconds. The limits on these three arguments are as follows (in seconds):

```
2 (hellotimer +1)  maxage   2 forwdelay -1)
```

- **maxage** *age*

  The maximum age (in hundredths of a second) that should be used to time out STP information. The range of *age* is 600-4000, and the default is 2000. This value takes effect only when this node becomes the root bridge.

- **forwdelay** *t*

  The interval (in hundredths of a second) before changing to another state. The value is in hundredths of a second, truncated to seconds, e.g. 400, 401, and 499 all signify 4 seconds. The range of *t* is 400-3000, and the default is 1500. This value takes effect only when this node becomes the root bridge.

- **hellotimer** *t*

  The interval (in hundredths of a second) between BPDUs sent out by this port. The value is in hundredths of a second, truncated to seconds, e.g. 400, 401, and 499 all signify 4 seconds. The range of *t* is 100-400, and the default is 200. This value takes effect only when this node becomes the root bridge.

- **pri** *pri*

  The priority for using this node vs. others for a path using the spanning tree protocol. The range of *pri* is 0-65535, and the default is 32768.

- **static** *MACaddr* **rcv** *rcv-port* **status** *arguments*

**static** *MACaddr* **rcv** *rcv-port* **xmit** *xmit-port* **[xmit-port . . .]**

Enter static entries into the bridge forwarding database.

— *MACaddr*

The MAC address to be used for forwarding, in the standard *xx*:*xx*:*xx*:*xx*:*xx*:*xx* format.

— **rcv {** *rcv-port* **| any }**

The port to which this MAC is assigned. The keyword **any** is a wildcard over all ports.

— **xmit** *xmit-port* **[***xmit-port* . . .**]**

The port to which received frames are to be forwarded.

— **status {invalid|deleteonreset|permanent}**

Set the status of the specified static bridge entry. The arguments are shown in Table 2-4:

**Table 2-4 status Arguments**

| Argument | Meaning |
|---|---|
| `invalid` | Delete the entry now. |
| `deleteonreset` | Delete the entry when the bridge is reset. |
| `permanent` | Do not delete the entry when the bridge is reset. |

# set tcs

For the specified card, set the midplane address, turn power on or off, or reset the card.

## Syntax

**set tcs** *card# attribute value* **| reset**

## Arguments

- **midplane** *address*

  Set the midplane address for the specified card to the node address *address*.

**Note** If you have a redundant system, you must change the midplane address for both switch A and switch B.

- **power {on|off}**

  Turn power on or off for the specified card.

- **reset**

  Reset the specified card.

## Description

**Note** The **set tcs** command requires CLI protected mode. (See the **protected** command.)

This command affects *only* the node on which the CLI is running when you execute it, regardless of a target set with the command **set snmp hostname** *name*.

# set trap

Control the display of specific traps.

## Description

See the *LightStream 2020 Administration Guide* for information about the using LightStream traps. See the *LightStream 2020 Traps Reference Manual* for information about individual traps.

## Syntax

**set trap {disable|enable}** *traps* **[***traps* **. . .]**

**set trap { [global] | pid {***number*|*name***} } {on|off}** *traps* **[***traps* **. . .]**

## Arguments

- *traps*

  Affect the trap identified by a trap name or a trap number. See the *Traps Reference Manual* for trap names and numbers. You may also specify a name for a group of traps as defined in the cli.groups file.

**Note** It is also possible to specify the *traps* parameter with a range of trap numbers. This is not recommended, as you could easily disrupt the system by flooding it with traps.

- **{disable|enable}**

  Disable specific traps that are being displayed on the console (normally, oper traps), or enable them again after they have been disabled. (This does not apply to traps that are being displayed because they were turned on with the **set trap on** command.)

- **{on|off}**

  Turn on traps that are currently being ignored (normally, trace traps and debug traps), or turn them back off again after they have been turned on. (This does not apply to traps that are being ignored because they were disabled with the **set trap disable** command.)

- **global**

  Make the **on** or **off** switch apply to all processes system wide. This is the default effect of **trap on** or **trap off**.

- **pid {***PIDname*|*PID#***}**

  Make the **on** or **off** switch apply to the process with alias *PIDname* or number *PID#*. You may display process aliases as follows:

```
cli> walksnmp lwmaTrapCliAlias
Name:lwmaTrapCliAlias.3 Value: IP
Name:lwmaTrapCliAlias.4 Value: RMON
Name:lwmaTrapCliAlias.5 Value: KLOG
Name:lwmaTrapCliAlias.6 Value: NPTMM
Name:lwmaTrapCliAlias.7 Value: COLLECTOR
Name:lwmaTrapCliAlias.8 Value: CAC
Name:lwmaTrapCliAlias.10 Value: GIDD
Name:lwmaTrapCliAlias.11 Value: LCC3
Name:lwmaTrapCliAlias.13 Value: NPCC
Name:lwmaTrapCliAlias.18 Value: LCC5
Name:lwmaTrapCliAlias.25 Value: LCC4
Name:lwmaTrapCliAlias.35 Value: ND
Name:lwmaTrapCliAlias.38 Value: TRAPMON
Name:lwmaTrapCliAlias.41 Value: lcmon (secondary)
Name:lwmaTrapCliAlias.47 Value: filterTask
cli>
```

---

**Note** The above description is based on the normal case, which is as follows:

---

- Oper traps are displayed on the console and on network management systems.
- Info and Oper traps are logged in the mma.traplog file.
- Trace and debug traps are ignored.

---

**Note** Although we do not recommend it, these norms may be reset with the following commands: **set chassis consoletraplevel**, **set chassis traplevel**, **set cli traplevel**, and **set pid** *pid#* **traplevel**. Doing this would change the ranges of traps affected by the **on**/**off** and **disable**/**enable** arguments.

---

---

**Note** The **set trap** command requires CLI protected mode. (See the **protected** command.) This command affects *only* the node on which the CLI is running when you execute it, regardless of a target set with the command **set snmp hostname** *name*.

---

## Examples

The following command sets the SNMP read/write community to write (a name to which the value write is assigned in the mma.communities file), so that you can use the **set trap** command:

cli> **set snmp community write**

The following command disables the oper trap LCC_1, which reports a high error rate:

cli> **set trap disable lcc_16**

After correcting the problem, re-enable the trap with the following command:

cli> **set trap enable lcc_16**

The following command sets the target system to be the node whose alias is boston:

cli> **set snmp hostname boston**

The following command makes port 5 on card 8 active:

cli> **set port 8.5 active**

# The show Command

Display the value of the specified MIB object(s).

## Description

The **show** command displays the value of the specified MIB object. The amount of information returned varies substantially, depending on the object.

## Syntax

**show** *type* **[***id***] [***parameter1* **[***parameter2***]]**

## Arguments

The *type* argument can be any of those shown in Table 2-5:

**Table 2-5 show Command type Arguments**

| *type* **Argument** | **Display** |
|---|---|
| **show bflt** | Bridge filters |
| **show card** | Per-card attributes |
| **show chassis** | Switch-wide attributes |
| **show cli** | CLI attributes |
| **show collection** | Collection records |
| **show config** | Status of configuration lock |
| **show gid** | Global information distribution (GID) attributes |
| **show modem** | Modem attributes |
| **show nd** | Neighborhood discovery (ND) attributes |
| **show pid** | Per-process attributes |
| **show port** | Per-port attributes |
| **show snmp** | SNMP control attributes |
| **show spt** | Spanning-tree bridge attributes |
| **show stb** | Spanning-tree bridge attributes |
| **show tcs** | Per-card attributes under TCS |
| **show trap** | State of traps |

The additional arguments that may be used with each *type* argument are described in the following pages.

# show bflt

Display bridge filter conditions that have been defined with the define command.

## Syntax

**show bflt [***ID***]**

## Argument

The optional *ID* argument limits the display to the specified filter; otherwise, all currently defined filters are displayed.

# show card

Display per-card attributes.

## Syntax

**show card** *card# parameter*

## Explanation

The *card#* argument is a card number. The *parameter* arguments are listed below.

## Arguments

- **all**

  Display all card attributes (name, process ID, status, version, hardware information, and information about configured ports). The other *parameter* arguments display selected parts of this display.

- **hardware**

  Displays the card type (e.g. LS edge), temperature readings at the top and bottom of the chassis, Voltages for TCS VCC, VCC, and Vee (or SCSI voltage for NP, BULK voltage for switch card), and temperature readings from region 1 and region 2 of an access card. The voltages are also displayed by the **show tcs** *card#* **voltage** command, where the display includes the normal voltage range for each. The command **show tcs** *card#* **temperature** also displays these temperature readings, where the display includes the warning and shutdown temperature values for each.

- **name**

  The card name (in the form *hostname.c*, where *c* is the card number).

- **ports**

  Displays information about each port configured on the card: the port number, the port type (e.g. Frame Relay), and the port name. The port number is displayed in long format, as described in the section “Port Identifiers.” The port name is in the form *hostname.c.p*, where *c* is the card number and *p* is the port number on that card.

- **processid**

The process ID associated with the specified card.

- **status**

  Displays the status of the card. Operational status is the actual current status of the card. Administrative status is the preferred status to which the card is restored on reset. Configuration register is the status of the board hardware, as read from the TCS board status register. In each case, the status may be up, down, testing, or empty (if no card is present in the slot).

- **version**

  Displays version information for LC software and LCC software on the card.

# show chassis

Display switch-wide attributes.

## Syntax

**show chassis** *parameter*

## Arguments

- **all**

  Display all chassis attributes (general, agent, congestion, primaryswitch, powersupply, cards, listff, listdlci, and listvci). The other *parameter* arguments display selected parts of this display.

- **agent**

  Display the following MMA attributes: MMA trap filter level, MMA trap logging state, MMA collection size, on/off state of the Config DB Active attribute, MMA PID number, configuration host name, configuration author, and configuration ID number.

- **cards**

  Display the type of card in each slot, or Empty if there is no card in a given slot.

- **congestion**

  Display the maximum and minimum intervals between permit limit updates, and the minimum interval between CA updates.

- **general**

  Display the host name, description (e.g. ATM Data Switch), contact person, location, system up time, console trap level, chassis ID, slot of NP being used by CLI, primary and secondary IP addresses and subnet mask, Ethernet address and mask, and default router address.

- **listff**

  Display a table of frame forwarding connections. The display shows the source node and port, the destination node and port, insured rate rate and bandwidth, and the maximum rate and bandwidth. If the VC is unconnected, the operationally requested maximum rate is displayed. If it is connected, the displayed value (which is actually in use) may have been negotiated down.

- **listdlci**

  Display a frame relay DLCI table for all frame forwarding ports. An asterisk (*) appears next to the port number if a circuit is deactivated. Compare the **show port** *c.p* **listdlci** command.

- **listvci**

  Display a VCI table for all ATM-UNI ports. Compare the **show port** *c.p* **listvci** command.

- **powersupply**

  Display the state and type of power supply A and powersupply B.

- **primaryswitch**

  Display which switch is primary (Switch A or Switch B).

# show cli

Display CLI attributes.

## Syntax

**show cli** *parameter*

## Arguments

- **all**

  Display all cli attributes (echosource, lineedit, log, term, time, timer, traplevel, debug, banner). The other *parameter* arguments display selected parts of this display.

- **banner**

  Display the banner that appears when you log in.

- **debug**

  Display the on/off setting of the debug switch (for LightStream development and support personnel only).

- **echosource**

  Display the on/off status of the CLI echo source. If echo source has been turned on with the command **set cli echosource on**, then the CLI commands in a script file are echoed to the screen when you execute them with the **source** command.

- **lineedit**

  Show whether command-line editing is on or off. This is controlled with **set cli lineedit**.

- **log**

  Display the status of the CLI trap logging function, either off or the name of the log file if it has been turned on with the command **set cli log** *logfile*.

- **term**

  Display the configured terminal type for the CLI, e.g. vt100.

- **time**

  Display the current date and time.

- **timer**

  Display the time elapsed since this CLI session was started, or since the command **set cli timer** was last executed.

- **traplevel**

  Display the current traplevel setting (Off, Oper, Info, Trace, or Debug). See the *LightStream 2020 Traps Reference Manual* for information about trap levels. See the *LightStream 2020 Administration Guide* for information about the using LightStream traps.

# show collection

Display collection record number collection#.

## Syntax

**show collection** *collection#*

## Description

The record includes collection status, operational status of the node, beginning and ending time of the collection, the interval between taking collection entries, and the pathname and size of the file used for the collection record. This is followed by the name and value of a series of collection items, e.g. the 17th object in collection 1 is collectDBObjectID.1.17, whose value is an entry for card 7, port 4 (ifInOctets.7004 or ifInErrors.7004).

# show config

Display status of configuration lock.

## Syntax

**show config**

## Description

This command displays the status (locked, unlocked) of the configuration database lock. See the **set config** command.

# show file

Display contents of log files.

## Syntax

**show file** *filetype* **[tail]**

The *filetype* argument of the **show file** command is not an actual file name, it is one of the following:

## Arguments

- **all**

  Display both of the log files (trap log and MMA log).

**Note** The **tail** argument is not available here.

- **collection** *collection#* **[tail]**

  Display the contents of the specified collection file.

- **mmalog [tail]**

  Display the contents of the MMA log file.

- **traplog [tail]**

  Display the contents of the trap log file.

### Description

With the **tail** parameter, display only the last 20 or so lines from the file.

**Note** This command accesses files *only* on the node on which the CLI is running when you execute it, regardless of a target set with the command **set snmp hostname** *name*. This command is not available when the CLI is run from a workstation.

## show gid

Display attributes of the global information distribution (GID) system.

### Syntax

**show gid** *parameter*

### Arguments

- **all**

  Display all gid attributes (general, flooding, synchronization, cards, clients, neighbors, ports, and ip). The other *parameter* arguments display selected parts of this display.

- **cards**

  Display a table of cards managed by GID, showing the host name (Chassis), slot number, sequence number, age (in seconds), originating NP, and number of configured ports.

- **clients**

  Display a table of clients managed by GID, showing for each client PID the number of link state (LS) announcements received, IP address translation announcements received, generic global information announcements received, transmitted events, and paths generated.

- **flooding**

  Display counts of flooding announcements received and transmitted.

- **general**

Display the software version number, the PID for GID, the amount of memory in use and the count of memory allocation failures for GID.

- **ip**

  Display a table of IP addresses managed by GID, showing for each IP address its age, sequence number, the advertising NP, network mask, and port.

- **neighbors**

  Display a table of neighbors managed by GID, showing for each host (chassis) name the VCI number, state, and counts of signals (SYNC, RLL, SLL, Hello, LSA, NLSA, IPA, GA, and NGA).

- **ports**

  Display a table of ports managed by GID, showing for each host (chassis) name and port the type of service, up/down state, counts for BW0, BW1, and BW2, and the remote port ID. BW0 is raw link capacity, BW1 is data bandwidth, and BW2 is bandwidth in use.

- **synchronization**

  Display counts of neighbors in the following states: existent sync, exchange start, exchange, loading sync, and full sync.

# show modem

Display the modem initialization string.

## Syntax

**show modem {SA|SB} {all|initstring}**

## Arguments

- **{SA|SB}**

  Specify the switch.

---

**Note** On the inactive switch, the CLI displays the display label Initstring: followed by a null.

---

- **all**

  Display the modem initialization string.

- **initstring**

  Display the modem initialization string.

## Description

This command affects *only* the node on which the CLI is running when you execute it, regardless of a target set with the command **set snmp hostname** *name*.

# show nd

Display attributes of the neighborhood discovery (ND) system.

## Syntax

**show nd** *parameter*

## Arguments

- **all**

  Display all ND attributes (general, ndcards, neighbors, switchupdown, switchstat, and clients). The other *parameter* arguments display selected parts of this display.

- **general**

  Display the software version number, PID for the ND, amount of memory in use by ND, and counts of timers processed, line cards managed by ND, neighbor NPs known to ND, and registered ND client processes.

- **ndcards**

  Display a table of cards managed by ND showing EIA port, channel number, and up/down state.

- **neighbors**

  Display a table of ND neighbors showing EIA port, channel number, and up/down state.

- **switchupdown**

  Display a table of the operational and administrative values of ND up/down parameters for each card, as follows:

| Parameter | Description |
|---|---|
| `interval` | The interval (in multiples of 100 ms) between up/down messages that ND sends to this card (default 300 ms). |
| `J` | ND must receive J/M messages to bring the line up (default 1). |
| `K` | ND must receive K/N messages to bring the line up (default 1). |
| `M` | ND must receive J/M messages to bring the line up (default 1). |
| `N` | ND must receive K/N messages to bring the line up (default 1). |

  These parameters may be used to fine-tune ND performance. They may be modified by setting the MIB objects ndAdminIntvl, ndAdminJ, etc., but this is not recommended.

- **switchstat**

  Display a table of ND switch statistics showing for each slot the number of switch up/down cells in and out, with error counts for each.

- **clients**

  Display a table of ND clients showing for each PID the type (e.g. sys, ca, gid, or lcc), subtype, and mask.

# show pid

Display per-process attributes for process pid#.

## Syntax

**show pid** *pid# parameter*

## Arguments

- **all**

  Display all per-process attributes (name, clialias, createtime, adminstatus, operstatus, traplevel) for process *pid#*. The other *parameter* arguments display selected parts of this display.

- **name**

  Display the PID name for process *pid#*.

- **clialias**

  Display the alias used by the CLI for process *pid#*.

- **createtime**

  Display the time elapsed (up time) since process *pid#* was created.

- **adminstatus**

  Display the administrative status for process *pid#*.

- **operstatus**

  Display the operational status of process *pid#*.

- **traplevel**

  Display the trap level set for process *pid#*. See the *LightStream 2020 Traps Reference Manual* for information about trap levels.

# show port

Display per-port attributes.

## Description

Use the **show port** command to display various attributes of a port. Most attributes can be displayed only for an appropriate card type. The types of attributes include the port state (for all card types), bridging and VLI attributes, and protocol-specific characteristics. These are described under the following headings:

- General Arguments of show port
- ATM VCI Arguments of show port
- Bridge Arguments of show port
- FDDI Arguments of show port
- Frame Forwarding Arguments of show port
- Frame Relay Arguments of show port
- VLI Arguments of show port

## Syntax

**show port** *c.p* **[***arguments***]**

## General Arguments of show port

- **all**

  Display all port attributes (name, status, statistics, physical, frameforward, framerelay, DLCI, and VCI). The other *parameter* arguments display selected parts of this display. This is the default, with **show port** *c.p* followed by no arguments.

---

**Note** Ethernet MAC addresses are canonical form (least significant bit first) and FDDI MAC addresses are displayed in MSB form (most significant bit first). So, for example, Ethernet MAC address 08:00:2B:3C:7F:E2 is displayed as FDDI MAC address 10:00:4D:3C:FE:47.

---

- **name**

  Display the description, name, type, MIB2 type, MTU, and speed for the specified port.

- **physical**

  Display some (depending upon port type) of the following physical attributes of the specified port: port type, operational and administrative CSU type, operational and administrative DCE receive bit rate, operational transmit bit rate, measured bit rate, link transmit utilization rate (data plus control), administrative expected dte rate and operational and administrative net interface type (dte/dce; these are for low-speed line cards only), operational and administrative protocol, LC auto enable state and debug level, data cell capacity and available capacity, call setup retry and backoff times, operational maximum frame size, and modem status (DCD, DSR)

- **status**

  Display the administrative status, operational status, and last change in operational status for the specified port.

- **statistics**

  Display statistical counts for the specified port.

## ATM VCI Arguments of show port

- **listvci**

  Display a list of ATM PVCs in order of their VCI numbers for the specified port. The VCI numbers are in column 3 of the display. If the two virtual connections that form the virtual circuit do not have the same values, they are displayed on two lines. A displayed (operational) value may differ from the requested value if it has been negotiated down. An asterisk at the beginning of the line means that circuit is down, or has not been created; an uppercase I means that the circuit has been set to inactive manually with the **set port** *c.p* **vci** *vci#***deactivate** command.

- **vci** *vci#*

  For the specified ATM-UNI port, display the following attributes of the PVC with the specified VCI: source node, port, and VCI; source insured rate, insured burst, maximum rate, and maximum burst (operational and administrative); destination operational node, port, VCI, insured rate, insured burst, maximum rate, and maximum burst; to-net and from-net circuit ID and circuit state, last error reported by ATMM, and cells required; counts of cells to the switch with CLP= 0 or 1, a count of cells to the switch with CLP = 0 upon arrival at the port, but forwarded with CLP = 1, and a count of discarded cells.

## Bridge Arguments of show port

- **show port** *c.p* **stb**

  Display the bridging state of the specified port (enabled or disabled for bridging), its priority (a value used in the Spanning Tree Protocol), and the port path cost (the contribution of this port to the path cost of those paths toward the root bridge that include it).

- **bcast-limit**

  Display the default broadcast rate limit that has been set for this port and the number of excess broadcast frames that have been dropped at this port as a result of applying this limit. Multicast is not supported in Release 2.0.

- **bflt [***ID***]**

  Display the filters that are associated with the specified port, plus some statistical information. The optional *ID* argument limits the display to the specified filter. The display includes the filter ID, priority, action, and a count of the number of times the filter was matched on this port.

- **bflt-def**

  Display the default bridging action that has been set for this port (**drop** or **forward)**.

## FDDI Arguments of show port

- **fddi** *arguments*

  Display FDDI information. Displayed values include: TVX (valid transmit timer), TReq (target request time), TRT (token rotation time), TNeg (negotiated target token rotation time or TTRT), T Max. (maximum token rotation time) value, RMT (ring management), UNDA (upstream neighbor duplicate address) flag, For the meanings of these and other displayed items, refer to the specification FDDI SMT X3T9.5 (Rev. 7.2). The *arguments* of the **show port** *c.p* **fddi** command are as follows:

  FDDI MAC addresses are displayed in MSB form (most significant bit first) and Ethernet MAC addresses are canonical form (least significant bit first). So, for example, Ethernet MAC address 08:00:2B:3C:7F:E2 is displayed as FDDI MAC address 10:00:4D:3C:FE:47.

  — **fddi mac**

    Display the following MAC information for this FDDI port: station management (SMT) index, MAC index, frame status functions, maximum capability, TVX capability, available paths, current path, upstream number, downstream number, old upstream number, old downstream number, duplicate address test, requested paths, downstream port type, SMT address, TReq, TNeg, T Max value, TVX value, frame count, copied count, transmit count, error count, lost count, frame error ratio, RMT state, duplicate address flag, UNDA flag, frame error flag, unit data present, hardware present, unit data enable flag.

  — **fddi maccounters**

    Display statistical information about FDDI traffic on this circuit. (The command is not case-sensitive, you may enter it as **maccounters**.) The items displayed are: MAC index, token count, TVX expirations, not copied count, TRT expirations, ring op count, not copied ratio, not copied flag.

  — **fddi path {1|2}**

    Display the following information for FDDI path 1 (primary path) or 2 (secondary path): the SMT index (calculated per port), path index (same as the path number), minimum TVX value, minimum TMax value, and maximum TReq value.

— **fddi {aport|bport}**

Display the following information about FDDI port A or FDDI port B in this FDDI circuit: station management (SMT) index, path index, PC type, remote type, connection policy, MAC indicated, current path, requested paths, MAC placement, available paths, PMD type, connection capabilities, BS flag, failed confidence test count, LER estimate, link error count, LER cutoff, LER alarm, connect state, PCM state, PC withhold, LER flag, hardware present, and action. See the command **set port** *c.p* **fddi {aport|bport}** for details about these fields.

— **fddi smt**

Display the following FDDI station management (SMT) information: the SMT index, Path index, Minimum TMax value, and Maximum TReq value.

## Frame Forwarding Arguments of show port

- **frameforward**

Display the following frame forwarding attributes of the specified port: source node and port, and destination node and port (operational and administrative); source and destination insured rate, insured burst, maximum rate, and maximum burst (operational and administrative for source, operational for destination); to-net and from-net circuit ID and circuit state, last error reported by ATMM, and number of cells required; counts of frames and cells to and from the switch with CLP= 0 or 1, and counts of discarded frames and cells. If the VC is unconnected, the operationally requested maximum rate is displayed. If it is connected, the displayed value (which is actually in use) may have been negotiated down.

## Frame Relay Arguments of show port

- **dlci** *dlci#*

Display the following attributes of the specified frame relay DLCI: source node, port, and DLCI; source insured rate, insured burst, maximum rate, and maximum burst (operational and administrative); destination node, port, and DLCI (operational and administrative); destination operational insured rate, insured burst, maximum rate, and maximum burst; local and remote link management interface (LMI) state, to-net and from-net circuit ID and circuit state, last error reported by ATMM, and cells required; counts of frames and cells to and from the switch with cell loss priority (CLP) = 0 or 1, and counts of discarded frames and cells.

---

**Note** Port number 1255 or 2255 refers to the control port (port 255) of the NP in slot 1 or slot 2, respectively.

---

- **framerelay**

Display the following frame relay attributes: the type of active LMI system, address length, request interval, the maximum for status query errors, the status query period, the maximum supported VCs, the type of broadcast supported (e.g. Uni-cast), the user polling interval, full enquiry interval, a count of user monitored events, and the net interface type (e.g. NNI).

- **listdlci**

Display a list of frame relay PVCs for the port in order of their DLCI numbers. The DLCI numbers are in column 3 of the display. If the values specified for the two directions of the circuit are not the same, they are displayed on two lines. A displayed (operational) value may differ from

the requested value if it has been negotiated down. An asterisk at the beginning of the line means that circuit is down, or has not been created; an uppercase I means that the circuit has been set to inactive manually with the **set port dlci** *dlci#***deactivate** command. Compare the **show chassis listdlci** command.

---

**Note** Port number 1255 or 2255 refers to the control port (port 255) of the NP in slot 1 or slot 2, respectively.

---

### VLI Arguments of show port

- **wgrp**

  Display the ID numbers of the workgroups associated with the specified port (if any), and the sense of the workgroup list (include or exclude).

---

**Note** Maintain a list of group IDs and the corresponding mnemonic workgroup names maintained by the configuration tool. The **show** command displays only the numeric IDs.

---

## show snmp

Display SNMP control attributes.

Display the read/write community name, the host name (target), and the on/off state of the authentication attribute.

---

**Note** These are all SNMP attributes.

---

### Syntax

**show snmp**

## show spt

Display spanning bridge parameters (listed below).

### Syntax

**show spt**

### Explanation

Display the following spanning-tree bridge parameters: designated root bridge address, root path cost, port for lowest cost path, protocol used (e.g. IEEE 8021d), maximum age, hello time, forward delay, priority, base bridge address, bridge max age, bridge hello time, bridge forward delay, topology change time, hold time, topology changes, bridge aging time, and the list of bridge ports with the spanning state of each.

Two values are displayed for the maximum age, hello time, and forward delay. One set of values is provided by the root bridge; the other set of values is established with the **set stb** command and takes effect only when the present node becomes the root bridge. These pairs of values are as follows:

| Provided by root bridge | Set with set stb |
|---|---|
| Bridge Max Age | Maximum age |
| Bridge Hello Time | Hello time |
| Bridge Forward Delay | Forward delay |

# show stb

Display the spanning-tree bridge parameters that have been set with **set stb** and with **set port** *c.p* **stb** commands.

## Syntax

**show stb {general|static|fwd|ports}**

## Arguments

- **general**

  Display the following bridge parameters: maximum age, hello timer, forward delay, and priority (for the node). The values displayed for Bridge Max Age, Bridge Hello Timer, and Bridge Forward Delay are those provided by the root bridge.

- **static**

  Display static entries in the bridge forwarding table (see **set stb static**), including the following entries: MAC address, receive port, and allowed transmit ports.

- **fwd**

  Display the bridge forwarding table with the following entries: MAC address, static/dynamic, chassis, interface (card, port).

- **ports**

  Display a list of all ports enabled for bridging, with the following parameters: port number, port state, received frames, forwarded frames, and dropped frames.

# show tcs

Display per-card attributes that are accessible using TCS.

## Description

This command affects *only* the node on which the CLI is running when you execute it, regardless of a target set with the command **set snmp hostname** *name*.

## Syntax

**show tcs** *card#* **{all|config|daughter|midplane|oem|paddle|power} [***argument***]**

## Arguments

- **all**

  Display all card attributes that are accessible using TCS (state, config, daughter, paddle, oem, midplane, temperature, voltage, and power). The other *parameter* arguments display selected parts of this display.

- **config** *parameter2*

  Display card configuration attributes that are accessible using TCS. With *parameter2* the **config** argument is as follows:

  — **config all**

  Display all the card configuration attributes. For explanation of the display, see the descriptions of the other *parameter2* arguments, below, which display selected parts of this display.

  — **config assembly**

  Display the assembly number (the manufacturing part number) of the board.

  — **config postcode**

  Display the revision number of the power-on self-test (POST) software on the board, not implemented in Release 2.0).

  — **config serialnum**

  Display the board serial number.

  — **config slavecode**

  Display the version number of the TCS slave on the board.

  — **config type**

  Display the card type, using abbreviations such as the following:

| Type | Description |
|---|---|
| S1 | Switch card 1 |
| S2 | Switch card 2 |
| N1 | Network processor (NP) card 1 |
| L1 | Low-speed line card 1 |
| M1 | Medium-speed line card 1 |
| P1 | Packet line card 1 |
| C1 | Cell line card 1 |

- **daughter** *parameter2*

  Display TCS daughter card attributes. With *parameter2* the **daughter** argument is as follows:

  — **daughter all**

  Display both TCS daughter card attributes (assembly and serial number). The other *parameter2* arguments each display just one part of this display.

  — **daughter assembly**

Display the daughter card assembly number (part number).

— **daughter serialnum**

Display the daughter card serial number.

- **midplane** *parameter2*

Display midplane attributes that are accessible using TCS. The *card#* argument must be **sa** or **sb**. With *parameter2* the **midplane** argument is as follows:

— **midplane all**

Display both TCS midplane attributes (assembly and serial number). The other *parameter2* arguments each display just one part of this display.

— **midplane assembly**

Display the midplane assembly number (part number).

— **midplane nodeaddress**

Display the midplane node address.

— **midplane serialnum**

Display the midplane serial number.

- **oem** *parameter2*

Display attributes for the portion of the access card that is reserved for development partners, which are accessible using TCS. With *parameter2* the **oem** argument is as follows:

— **oem all**

Display both TCS access card attributes (assembly and serial number). The other *parameter2* arguments each display just one part of this display.

— **oem assembly**

Display the access card assembly number (part number).

— **oem serialnum**

Display the access card serial number.

- **paddle** *parameter2*

Display attributes for the LightStream portion of the access (paddle) card, which are accessible using TCS. With *parameter2* the **paddle** argument is as follows:

— **paddle all**

Display both TCS access (paddle) card attributes (assembly and serial number). The other *parameter2* arguments each display just one part of this display.

— **paddle assembly**

Display the access (paddle) card assembly number (part number).

— **paddle serialnum**

Display the access (paddle) card serial number.

- **power**

Display on/off power state of the specified card.

- **state**

Display the status of the following card attributes that are accessible using TCS power supply, temperature, clock, POST, XILINX load, application load, access (paddle) cards and overrides, flash memory, CP POST, application software, card, power (TCS VCC, VCC, VPP, and SCSI), temperature (top and bottom), board initialization, flash initialization, and TCS HUB.

- **temperature**

  Display temperature readings for the top and bottom regions of the specified card, which are accessible using TCS. The command **show card** *card#* **hardware** also displays these temperature readings.

- **voltage**

  Display voltage readings and normal voltage ranges for TCS VCC, VCC, VEE (or SCSI for NP), and VPP voltage (except for switch card) on the specified card, which are accessible using TCS. The command **show card** *card#* **hardware** also displays these voltages.

# show trap

Display state of traps, either system-wide (the default) or by process.

## Syntax

**show trap {[global] | pid {***pid#|pidname***}}** *trapspec*

**[ {[global] | pid {***pid#|pidname***}}** *trapspec* **. . .]**

## Arguments

One of the following optional arguments may be used to filter traps by their origination:

- **pid {***PID#|PIDname***}**

  Display traps from the process whose process ID is *PID#* or whose alias is *PIDname*.

- **global**

  Display traps from all processes, system wide. This is the default.

The *trapspec* arguments are used to specify which traps to display, as follows:

- *trapname*

  The trap identified by the name *trapname*. (See the *Traps Reference Manual*.)

- *trapgroup*

  The set of traps identified by the definition of the name *trapgroup* in the cli.groups file.

- *trap#*

  The trap identified by *trap#*.

- *trap#-trap#*

  The set of traps identified the range of traps *trap#-trap#*.

## Description

See the *LightStream 2020 Administration Guide* for information about the using LightStream traps. See the *LightStream 2020 Traps Reference Manual* for information about trap levels and individual traps.

**Note** This command affects *only* the node on which the CLI is running when you execute it, regardless of a target set with the command **set snmp hostname** *name*.

## Examples

The following example shows how to determine what cards are configured on the switch that you are interrogating:

```
cli> show chassis cards
Slot 1: NP
Slot 2: LS Trunk
Slot 3: LS Edge
Slot 4: LS Edge
Slot 5: MS Trunk
Slot 6: ATM-UNI
Slot 7: Empty
Slot 8: Empty
Slot 9: Empty
Slot 10: LS Edge
Slot SA: Switch
Slot SB: Empty

cli>
```

The following example shows how to determine what ports are configured on card 3 on the switch that you are interrogating:

```
cli> show card 3 ports
Port 3000 Frame Relay Name: Blue Dollar - 4
Port 3001 Frame Relay Name: Blue Dollar - 5
Port 3002 Frame Relay Name: Blue Dollar - 8
Port 3003 Frame Relay Name: Blue Dollar - 9
Port 3004 Frame Relay Name: lstb7.3.4_fr
Port 3005 Frame Relay Name: Purple Plus - 1
Port 3006 Frame Forwarding Name: Green Asterisk - 8
Port 3007 Frame Forwarding Name: Green Asterisk - 9
cli>
```

The following example shows how to display port statistics for port 6 on card 3 in the switch that you are interrogating:

```
cli> show port 3.6 statistics
Octets Rcvd: 3356927738
Normal Packets Rcvd: 83574214
Multicast Packets Rcvd: 0
Discarded Rcvd Packets: 31524
Receive Errors: 4
Unknown Protocols Rcvd: 0
Octets Sent: 4070237100
Normal Packets Sent: 46399345
Multicast Packets Sent: 0
Discarded Output Packets: 0
Output Errors: 3

cli>
```

The following example shows how to identify the daughter board serial number for the TCS on card 3:

```
cli> show tcs 3 daughter serialnum
Slot 3 Daughter Serialnum: SM332-18
cli>
```

# CLI Control Commands

This section lists the commands that you use to monitor and control the CLI session. It also includes the **ping** command, which may be used with the **set** and **show** commands, described in the sections "The set Command" and "The show Command," to monitor and control the LightStream node from the CLI.

| Type | Name | Function |
|---|---|---|
| CLI Control Commands | **clear** | Clear the screen. |
| | **exit** | Exit or protected mode or CLI program (same as **quit**). |
| | **help** | Display online help for CLI commands. |
| | **password** | Change the password for protected mode. |
| | **ping** | Send ICMP echo packets to a host and report on any returned packets. |
| | **protected** | Allow access to protected mode. |
| | **quit** | Exit CLI or protected mode (same as **exit**). |
| | **shell** | Execute a LynxOS command under a copy of the LynxOS shell. |
| | **source** | Execute a CLI script (CLI commands stored in a disk file). |

## clear

Clear the screen.

Use the **clear** command to clear the screen.

### Syntax

**clear**

## exit

Exit protected mode or the CLI program.

### Syntax

**exit**

### Arguments

None.

### Description

In normal mode, the **exit** command halts the CLI program and returns you to the bash> prompt. If you are in protected mode, the **exit** command returns you to normal mode in the CLI, so that you must type **exit** a second time to halt the CLI.

**Note** The **exit** command provides the same functions as the **quit** command.

### Example

```
*cli> exit
cli> exit
bash>
```

## help

Provide online help for CLI commands.

### Syntax

**help [***topic***]**

### Arguments

**Note** The *topic* argument is optional.

- *topic*

  Specifies the name of the command on which you want help.

  By default (if you do not enter a *topic* argument), or if you enter the command **help help** or **help ?**, the **help** command displays a list of all CLI commands.

### Description

The **help** command displays online help information about CLI commands.

Refer to Chapter 4 of the *LightStream 2020 Operations Guide* for a detailed description of the online help facility in the CLI.

### Example

The following example shows the display that the **help** command displays with **protected** as its argument:

```
cli> help protected
NAME

protected - enter protected mode

SYNTAX


protected

DESCRIPTION
```

```
Enter protected mode. User will be prompted for a password.
Password will not be echoed as it is entered. Protected mode
is required for sensitive or potentially disruptive operations.

cli>
```

# password

Change the password for the npadmin account, which is also the password for protected mode. Requires protected mode.

## Syntax

**password**

## Arguments

N/A

## Description

Use the **password** command to change the password for the npadmin account, which is also the password for accessing protected mode in the CLI:

---

**Note** There are four accounts on the NP: oper, npadmin, root, and fldsup. To change the password for any of these accounts, use the LynxOS **passwd** command (note the difference in spelling), as described in Chapter 3 of the *LightStream 2020 Operations Guide*. There are thus two ways to change the npadmin password (which is also the password for protected mode), but only one way to change the root, oper, or fldsup password.

---

---

**Note** This command affects *only* the node on which the CLI is running when you execute it, regardless of any target set with the command **set snmp hostname** *name*.

---

## Example

```
*cli> password
Changing NIS password for npadmin on Boston5.
Old password:
New password:
Retype new password:
NIS entry changed on Boston5
*cli>
```

# ping

Send ICMP echo packets to a host and report on any returned packets.

## Syntax

**ping [***packet-size***]** *host-name*

## Arguments

- *packet-size*

  Specifies the size of the packets that are sent.

**Note** The *packet-size* argument is optional, and the default packet size is 64 bytes.

- *hostname*

  Specifies the name of the host to which to send packets. The *hostname* argument is an IP address specified either by name (e.g. nnsc.nsf.net or boston5) or by number (e.g. 197.5.0.8 or 198.113.178.17). See Chapter 4 of the *LightStream 2020 Administration Guide* for information about creating the hosts file that defines host names.

## Description

The **ping** command sends a series of ICMP echo packets at 1-second intervals to the specified IP address and reports on any returned ICMP echo-response packets. Press **^C** (the control-C key) to stop the ping command and display a summary of the results.

**Note** Because of the load it could impose on the network, it is unwise to use **ping** during normal operations or from automated scripts.

**Note** This command affects *only* the node on which the CLI is running when you execute it, regardless of a target set with the command **set snmp hostname** *name*.

## Example

The following example shows four ping packets sent to the node named boston5:

```
cli> ping boston5
PING boston5 (198.113.178.17): 64 data bytes
64 bytes from 198.113.178.17: icmp_seq=0 time=26 ms
64 bytes from 198.113.178.17: icmp_seq=1 time=12 ms
64 bytes from 198.113.178.17: icmp_seq=2 time=13 ms
64 bytes from 198.113.178.17: icmp_seq=3 time=14 ms
^C

----boston5 PING Statistics----
4 packets transmitted, 4 packets received, 0packet loss
round-trip (ms) min/avg/max = 12/17/26


cli>
```

The IP address 198.113.178.17 could have been used as the command argument instead of the alias boston5. The command used the default packet size of 64 bytes. The example shows the number of packets transmitted, the number received back from boston5, and the minimum, average, and maximum round-trip transmission time.

# protected

Allow access to protected mode.

## Syntax

**protected**

## Arguments

None.

## Description

The protected command is used to access the CLI in protected mode. When you are in protected mode, you have access to additional commands. The command prompts you for the password for protected mode. You cannot enter protected mode unless you enter the correct password. When you are operating in protected mode, the cli> prompt is preceded by an asterisk (*cli>). To terminate protected mode and return to the cli> prompt, use the exit or quit command.

**Note** The password for protected mode is the same as the password for the npadmin account.

## Example

To enter protected mode, enter the **protected** command. You are prompted for the password for the protected mode. When you enter the correct password and press **[Return]**, you enter protected mode. In protected mode, the cli> prompt is always preceded by an asterisk, *cli>.

```
cli> protected
Enter Password:
*cli>
```

# quit

Exit protected mode or the CLI program.

## Syntax

**quit**

## Arguments

None.

## Description

The **quit** command halts the CLI program. If you are in protected mode, the quit command exits from protected mode, but remains in the CLI (in normal mode), so that you must type **quit** a second time to exit from the CLI.

**Note** The **quit** command provides the same functions as the **exit** command.

## Example

```
*cli> quit
cli> quit
```

# shell

Execute a LynxOS command under a copy of the LynxOS shell. Requires protected mode.

## Syntax

**shell "***LynxOS command***"**

## Arguments

- *LynxOS command*

  Any LynxOS command that you can normally execute from the bash shell. Place the command in quotation marks.

## Description

The **shell** command executes a LynxOS command as if it were being run from the bash shell. The actual LynxOS command must be surrounded by quotes. For additional information about LynxOS commands, see the "LynxOS Command Reference" chapter. If more than one command is to be executed, it is often more convenient to start a subshell with the command **shell bash** at the cli> prompt. If you do this, it is very important to terminate the subshell afterward with the shell built-in command **exit** (see the "BASH Shell Reference" appendix).

**Note** This command affects *only* the node on which the CLI is running when you execute it, regardless of a target set with the command **set snmp hostname** *name*.

## Example

The following example shows use of the shell command to run a LynxOS command that displays the date and time as known to the system. Because the command is a single word, the quotation marks are optional:

```
*cli> shell date
Thu Feb 17 13:39:52 EST 1994
*cli>
```

The following example shows use of the **ls** command to display names of all files that begin with a dot (.). Note that **ls .*** or **ls .?*** would display contents of the parent directory (abbreviated .. by the shell). Because the command contains a space, it must be entered within quotation marks:

```
*cli> shell ls ".??*"
.bash_history .profile .rhosts .rhosts.dist
*cli>
```

The following example shows use of the **cbufpr** command to display the last lines of the mma.traplog file. The command contains spaces, so it must be enclosed in quotation marks:

```
*cli> shell "cbufpr -t /usr/tmp/mma/mma.traplog"
PROGRAM: cbufpr: compiled Feb 02 1994 @ 16:07:57 [pid:35]
(OPER) NDD_3 at 02/17/94 16:35:29 EST (02/17/94 21:35:29 GMT)
Line Card lstb4:3 (MS-TR) up.
(INFO) NPTMM_2007 at 02/17/94 16:35:30 EST (02/17/94 21:35:30 GMT)
Slot 3 State Changed From DOWN To UP
(INFO) LC_2000 at 02/17/94 16:35:30 EST (02/17/94 21:35:30 GMT)
Slot 3: ND: ERMP channel to NP open
(INFO) LC_2000 at 02/17/94 16:35:38 EST (02/17/94 21:35:38 GMT)
Slot 6: FIRMWARE: CMP 0x12 PCP 0x8 FSU 0x10201
(INFO) LC_2000 at 02/17/94 16:35:39 EST (02/17/94 21:35:39 GMT)
Slot 6: LC 5148.6 booting: flash_status OK idata_image OK
(INFO) LC_2000 at 02/17/94 16:35:39 EST (02/17/94 21:35:39 GMT)
Slot 6: ports enabled 0x00
(INFO) LC_2000 at 02/17/94 16:35:39 EST (02/17/94 21:35:39 GMT)
Slot 8: FIRMWARE: CMP 0x12 PCP 0x6 FSU 0x10201
(INFO) LC_2000 at 02/17/94 16:35:40 EST (02/17/94 21:35:40 GMT)
Slot 8: LC 5148.8 booting: flash_status OK idata_image OK
(INFO) LC_2000 at 02/17/94 16:35:40 EST (02/17/94 21:35:40 GMT)
Slot 8: ports enabled 0x00
(OPER) NDD_3 at 02/17/94 16:35:46 EST (02/17/94 21:35:46 GMT)
Line Card lstb4:6 (LS-EDGE) up.
*cli>
```

The following example shows use of the command **bash** to start a subshell for purpose of executing multiple commands. It then shows use of the shell built-in command **exit** to return to the CLI:

```
*cli> shell bash
bash$ cd /usr/app/base/config
bash$ vi cli.groups
```

**Note** The displays from the **vi** editing session are omitted from this example.

```
bash$ exit
exit
*cli>
```

The effects of the **cd** command are lost when you terminate the subshell. The next use of the **shell** command will execute LynxOS commands in the user's login directory.

## source

Execute a CLI script (CLI commands stored in a disk file).

## Syntax

**source "***filename***"**

## Arguments

- *"filename"*

  Identifies the file that contains the CLI commands that you want to execute. The file *filename* must be in the current directory (usually the same directory as the user account you are using), or in the /etc, /bin, or /config directory. If it is not, you must enter the full pathname of the file.

**Note** Always surround the file name or pathname with quotation marks.

## Description

The **source** command executes CLI commands that are stored in a disk file (a script), each command beginning on a new line. Comments may be included with the CLI commands in C programming style, between an initial **/*** string and a final ***/** string.

**Note** This command affects *only* the node on which the CLI is running when you execute it, regardless of a target set with the command **set snmp hostname** *name*.

## Examples

```
cli> source "show_all_ports"
cli> source "/usr/tmp/mma/show_recent_traps"
cli>
```

# MIB Commands

This section describes commands used to display and modify the values of MIB objects (see the "LightStream MIB Reference" chapter).

| Type | Name | Function |
|---|---|---|
| MIB Commands | **browse** | Browse the MIB tree. |
| | **getsnmp** | Display the value of a MIB object. |
| | **getnextsnmp** | Display the value of the object in the MIB tree that follows the specified object. |
| | **setsnmp** | Change the state of the specified MIB object. |
| | **walksnmp** | Display the values of all MIB objects in the MIB tree starting with the specified object. |

# browse

Browse the MIB tree.

## Description

Use the **browse** command to walk through the MIB tree and obtain the value of any object in the MIB tree.

## Syntax

**browse [***mib-address***]**

## Arguments

**Note** The *mib-address* argument of this command is optional.

- *mib-address*

  Specifies the point in the MIB tree at which the browse tool opens. If you do not specify this argument, the browse tool opens at the top of the MIB tree (at the iso object).

  The address can be specified as a path of dot-separated numbers, or as a variable name, or a combination (where only the first element is a variable name). Thus, 1.3.6.1.2.1, mgmt.1, and mib all refer to the same variable. (See the section "Alternative Forms of MIB Addresses" for a description of alternative forms of MIB addresses.)

## Examples

The following example shows how to start the **browse** command with the address of a particular object, *mgmt* in this example:

```
cli> browse mgmt

mgmt:
1) mib
Enter line number to go down, 'u' to go up, 'q' or 'e' to quit browse.
browse>
```

There is only one object under the mgmt object, so we select it by entering the menu number 1:

```
browse> 1

mgmt.mib:
1) system
2) interfaces
3) at
4) ip
5) icmp
6) tcp
7) udp
8) egp
9) transmission
10) snmp
11) dot1dBridge
12) rmon
Enter line number to go down, 'u' to go up, 'q' or 'e' to quit browse.
browse>
```

There are 12 objects under the mib object. We select the object *system* by entering the menu number 1:

```
browse> 1

mgmt.mib.system:
1) sysDescr
2) sysObjectID
3) sysUpTime
4) sysContact
5) sysName
6) sysLocation
7) sysServices
Enter line number to go down, 'u' to go up, 'q' or 'e' to quit browse.
browse>
```

All of the objects under the system object are "leaves" of the MIB tree, that is, there are no further branches under them. When we select one from the menu, the program displays the value for that object and returns to the mgmt.mib.system menu:

```
browse> 4
-----------------------------------------------------
Name: sysContact.0 Value: Lisa Bloch
mgmt.mib.system:
1) sysDescr
2) sysObjectID
3) sysUpTime
4) sysContact
5) sysName
6) sysLocation
7) sysServices
Enter line number to go down, 'u' to go up, 'q' or 'e' to quit browse.
browse>
```

If we wish to explore a different branch of the mib subtree, we can enter "u" to return to the next higher level:

```
browse> u

mgmt.mib:
1) system
2) interfaces
3) at
4) ip
5) icmp
6) tcp
7) udp
8) egp
9) transmission
10) snmp
11) dot1dBridge
12) rmon
Enter line number to go down, 'u' to go up, 'q' or 'e' to quit browse.
browse>
```

To exit to the cli> prompt, we enter "q" (or "e"):

```
browse> q
Leaving browse
cli>
```

# getsnmp

Display the value of a specified MIB object.

## Description

Given the addresses of one or more MIB objects, **getsnmp** displays the value of the MIB object at each address.

## Syntax

**getsnmp** *mib-address* **[***mib-address* **[ . . .***mib-address***]]**

## Arguments

**Note** Any *mib-address* arguments after the first are optional.

- *mib-address*

  Specifies the address of a MIB object that you want to display. As an optional, repeated argument, it specifies the address of an additional MIB object that you want to display.

  The address can be a path of dot-separated numbers, a variable name, or a combination (where only the first element is a variable name). Thus, 1.3.6.1.2.1.1.1.0, mib.1.1.0, system.1.0 and sysDescr.0 all refer to the same variable.

**Note** The address must end with a numeric qualifier: .0 in the case of an isolated leaf, a longer number in the case of a table entry, for example .4002 for slot 4 port 2.

If you want to get more than one object, you can list more than one object identifier. You can use the **walksnmp** command to identify MIB objects and their addresses.

## Examples

```
cli> getsnmp sysContact.0 sysServices.0
Name: sysContact.0 Value: Lisa Bloch
Name: sysServices.0 Value: 78
cli> getsnmp 1.3.6.1.2.1.1.4.0 mgmt.1.1.7.0
Name: sysContact.0 Value: Lisa Bloch
Name: sysServices.0 Value: 78
cli> getsnmp system.4.0
Name: sysContact.0 Value: Lisa Bloch
```

# getnextsnmp

Display the value of the next object after the specified object in the MIB tree.

## Description

Given the address of one or more MIB objects, **getnextsnmp** displays the value of the MIB object that comes next in the subtree after each address you specified. If you specify the last variable ("leaf" object) in a subtree, the command displays the first variable in the next following subtree.

## Syntax

**getnextsnmp** *mib-address* **[***mib-address* **[ . . .** *mib-address***]]**

## Arguments

**Note** Any *mib-address* arguments after the first are optional.

- *mib-address*

  Specifies the address of the MIB object just before the object that you want to display. As an optional, repeated argument, it specifies the address of an additional MIB object whose next-following neighbor in the MIB tree you want to display.

  The address can be specified as a path of dot-separated numbers, or as a variable name, or a combination (where only the first element is a variable name). Thus, 1.3.6.1.2.1.1.1, mib.1.1, system.1 and sysDescr all refer to the same variable.

  If you want to get more than one object, you can list more than one object identifier. You can use the walksnmp command to identify MIB objects and their addresses.

## Examples

```
cli> getnextsnmp sysContact.0 sysServices.0
Name: sysName.0 Value: Comet
Name: ifNumber.0 Value: 10007
cli> getnextsnmp 1.3.6.1.2.1.1.4.0 mgmt.1.1.7.0
Name: sysName.0 Value: lstb7
Name: ifNumber.0 Value: 10007
cli>
```

# setsnmp

Change the value of the specified MIB object. Requires protected mode.

## Description

Use **setsnmp** to set the value of a specified MIB object.

**Note** The **setsnmp** command requires that the read/write community name be set first to a name that has been assigned the value write in the mma.communities file. If the read/write community name has not been set first, the **setsnmp** command fails, because the default community name "public" is read only. See the description of the command **set snmp community** and Chapter 6 of the *LightStream 2020 Operations Guide* for information on setting the read/write community.

**Caution** Do not manipulate MIB objects unless you are familiar with SNMP.

## Syntax

**setsnmp** *MIBaddress value*

## Arguments

- *MIBaddress*

  Identifies the MIB variable to be set.

  The address can be a path of dot-separated numbers, or a variable name, or a combination (where only the first element is a variable name). Thus, 1.3.6.1.2.1.1.1.0, mib.1.1.0, system.1.0 and sysDescr.0 all refer to the same variable. You can use the walksnmp command to identify MIB objects and their addresses.

**Note** The address must end with a numeric qualifier: .0 in the case of an isolated leaf, a longer number in the case of a table entry, e.g. .4002 for slot 4 port 2.

- *value*

  Specifies the new value for the MIB object at *MIBaddress*. If the *value* argument contains spaces, you must quote it. The value may be a number, a string, or an object ID in the form **:***MIBaddress*. To determine the appropriate argument type for the specified MIB object, type ? after the first *MIBaddress* argument.

## Example

The following example shows use of **setsnmp** to change the name of the contact person for the system (note quotation marks):

```
*cli> setsnmp sysContact.0 "Tom Smith"
Name: sysContact.0 Value: Tom Smith
*cli>
```

The following example shows use of the online help facility (typing ? after a partial command) with **setsnmp** to determine the type of value that can be assigned to a given object:

```
*cli> setsnmp ifDescr.1000 ?
Enter an octet string
*cli> setsnmp ifDescr.1000
```

# walksnmp

Display the values of all MIB objects in the MIB tree starting with the specified object.

## Description

The **walksnmp** command displays the names and values of all variables that are "leaves" of the MIB tree below the specified MIB object. If you specify the head of a particular subtree (as shown in the figures in the "LightStream MIB Reference" chapter), the command displays all the "leaf" variables at the ends of branches in that subtree. This is equivalent to repeated getnextsnmp. If you specify an object that has no branches under it, it displays the value of that object, just like getsnmp.

Use the **walksnmp** command to survey a range of variables, to locate a variable quickly when you know only which part of the MIB it is in, or to identify the name of a MIB variable so that you can specify it as argument of another SNMP command.

## Syntax

**walksnmp** *mib-address*

## Arguments

- *mib-address*

  The starting point. The address can be specified by a path of dot-separated numbers, by a variable name, or by a combination (where only the first element is a variable name). Thus, 1.3.6.1.2.1.1, mgmt.1.1, and system all refer to the same subtree.

## Example

```
cli> walksnmp system
Name: sysDescr.0 Value: LightStream Data Switch
Name: sysObjectID.0 Value: lightStreamATM
Name: sysUpTime.0 Value: 26422638
Name: sysContact.0 Value: Lisa Bloch
Name: sysName.0 Value: Comet
Name: sysLocation.0 Value: Boston, 27/412
Name: sysServices.0 Value: 78
cli>
```

# VLI Commands

This section describes two commands used for creating and deleting bridge filters.

| Type | Name | Function |
|---|---|---|
| VLI Commands | **define** | Define a bridge filter. |
| | **delete** | Delete a bridge filter. |

See also the following commands:

| Name | Function |
|---|---|
| **set port** *c.p* **bcast-limit** | Set bridge filter parameters. See the section “set port.” |
| **set port** *c.p* **bflt** | |
| **set port** *c.p* **bflt-def** | |
| **show port** *c.p* **bcast-limit** | Display bridge filter parameters. See the section “show port.” |
| **show port** *c.p* **bflt** | |
| **show port** *c.p* **bflt-def** | |
| **set port** *c.p* **wgrp** | Set VLI workgroup parameters. See the section “set port.” |
| **show port** *c.p* **wgrp** | Display VLI workgroup parameters. See the section “show port.” |
| **show bflt** | Display bridge filter conditions and other chassis-wide bridge parameters. |
| **set stb** | Set spanning-tree bridge parameters. |
| **set port** *c.p* **stb** | Set spanning-tree bridge parameters. See the section “set port.” |
| **show stb** | Display spanning-tree bridge parameters. |
| **show spt** | Display spanning-tree bridge parameters. |

# define

Define a bridge filter.

## Description

The **define** command defines a filter condition that determines the bridge ports to which traffic is forwarded.

A filter is a condition or a set of conditions that determine whether incoming frames are blocked or forwarded. If the contents of a field in an incoming frame header match the value specified for that field in a filter condition, then a specified action is taken with that frame.

- Use the command **define bflt** *ID expression* to associate a filter ID with a specific set of filter conditions.
- Use the command **set port** *c.p* **bflt** *ID priority action* to specify the action to take if a filter is matched, where the *priority* argument determines the sequence in which filter conditions are considered for the specified port.
- Use the command **show bflt [***ID***]** to display currently defined filters.
- Use the command **show port** *c.p* **bflt [***ID***]** to display filters associated with a specified port, plus statistical information.
- Use the command **set port** *c.p* **bflt** *ID* **delete** to break the association of a filter with a port.
- Use the command **delete bflt** *ID* to delete a filter after all of its associations with ports have been broken.

A maximum of 512 filters may be defined.

## Syntax

**define bflt** *ID expression*

## Arguments

- **bflt**

  Specifies that it is a bridge filter that is to be defined. In the current release, this is the only object type that can be defined with this command.

- *ID*

  A number in the range 1-64000. If *ID* is already in use for a filter, the command asks if you want to overwrite the old filter, and lets you assign a different number if you answer no. We recommend that you assign *ID* values by 10s (i.e. 10, 20, 30, etc.), leaving ample gaps for future assignments. Use the **show bflt [***ID***]** command to display currently defined filters.

- *expression*

  A comparison expression or a boolean expression. In a comparison expression, the value of an incoming header field is compared with a constant. In a boolean expression, two or more expressions are combined with boolean operators.

  A comparison expression has one of the following two forms:

  *field operator constant*

  **(***field* **&** *mask***)** *operator constant*

  The syntax variables *field*, *mask*, *operator*, and *constant* are as follows:

  — *field*

  A built-in identifier for a field in incoming frame headers. The field identifiers are not case sensitive (e.g. macsrc is equivalent to macSrc). They are listed below.

| Field | Description | Field | Description |
|---|---|---|---|
| macSrc | MAC source address | llcDSAP | LLC destination SAP |
| macDst | MAC dest. address | llcCtl | LLC control |
| macProto | MAC protocol type | snapOUI | SNAP OUI |
| llcSSAP | LLC source SAP | snapProto | SNAP protocol |

  — *mask*

  The *field* may be masked with a C-style bitwise AND operator in an expression of the form (*field* **&** *mask*). Thus, the following expression ignores the final hex digit of the macEther field and matches a zero in the corresponding position of the *constant* argument:

```
(macEther & fff0) operator constant
```

  — *operator*

  A comparison operator, as follows:

| Operator | Meaning |
|---|---|
| == | equal |
| != | not equal |
| > | greater than |
| >= | greater or equal |
| < | less than |
| <= | less or equal |

— *constant*

The *constant* must be of the appropriate form for the built-in *field* on the left side of a simple comparison *expression*. They are all sequences of hex digits *x* (with leading zeroes if necessary, but without leading 0x), as follows:

| Field | Format | Description |
|---|---|---|
| macSrc | *xx:xx:xx:xx:xx:xx* | MAC source address |
| macDst | *xx:xx:xx:xx:xx:xx* | MAC destination address |
| macProto | *xxxx* | MAC protocol type |
| llcSSAP | *xx* | LLC source SAP |
| llcDSAP | *xx* | LLC destination SAP |
| snapOUI | *xxxxxx* | SNAP OUI |
| snapProto | *xxxx* | SNAP protocol |

**Note** The names of fields are not case sensitive in arguments of CLI commands. See the Glossary for the terms LLC, MAC, OUI, SAP, and SNAP.

A boolean expression results from conjoining expressions with boolean operators, as follows:

*expression boolean-operator expression* [...]

Here, *boolean-operator* is **&&** (logical AND) or || (logical OR), and *expression* may be either a boolean expression or a comparison expression. Evaluation of expressions is left to right. Parenthesized expressions are resolved first, so you may use parentheses to force a different evaluation sequence.

## Examples

```
cli> define bflt 10 llcCtl == 02
cli> define bflt 20 llcSSAP >= 1
cli> define bflt 30 macSrc == 00:dd:00:00:00:12 && macDst != \ 00:dd:00::0000:76
cli> define bflt 40 (macSrc & ff:ff:00:00:00:00) == 00:dd:00:00:00:00
cli> define bflt 50 (macSrc & ff:ff:00:00:00:00) == 00:dd:00:00:00:00 \ && (macDst &
ff:ff:00:00:00:00) != 00:dd:00:00:00:00
cli>
```

# delete

Delete a bridge filter.

## Syntax

**delete bflt** *ID*

## Arguments

- **bflt**

Specifies that it is a bridge filter that is to be deleted. In the current release, this is the only VLI object type that can be deleted.

- *ID*

  A filter ID, previously set with the **define bflt** *ID expression* command.

## Description

Delete a bridge filter condition previously set with a **define** command. If the filter has been assigned to a port with the command **set port** *c.p* **bflt** *ID priority action*, it cannot be deleted until the association is broken with the **set port** *c.p* **bflt** *ID* **delete** command.

- Use the command **define bflt** *ID expression* to associate a filter ID with a specific set of filter conditions.
- Use the command **show bflt [***ID***]** to display all currently defined filters.
- Use the command **show port** *c.p* **bflt [***ID***]** to display currently defined filters on a per-port basis.
- Use the command **set port** *c.p* **bflt** *ID priority action* to assign a filter to a port and to specify the sequence in which filter conditions are considered for the specified port and the action to take if a filter is matched.
- Use the command **set port** *c.p* **bflt** *ID* **delete** to break the association of a filter with a port.

# Diagnostics Commands

This section describes the following diagnostics and troubleshooting commands:

| Type | Name | Function |
|---|---|---|
| Diagnostics Commands | **connect** | Logically attach the console or modem I/O ports to a given card within a LightStream node. |
| | **loadcard** | Load the specified file into the specified card, start the card, and establish a console connection between the CLI and the TCS slave on the card. |
| | **test** | Run field diagnostics tests from the CLI. |
| | **read** | Read memory and hardware registers accessible by the TCS. For qualified personnel only. |
| | **write** | Write hex values to memory and hardware registers. For qualified personnel only. |

# connect

Connect the CLI to a line card as a console. Requires protected mode.

## Description

Use the **connect** command to connect the CLI to the specified line card so it can act as a terminal for a program, such as diagnostics, running on the line card.

Type ~. (tilde-dot) to interrupt the connection. When the **connect** command is used to reconnect a previously existing connection, it prints outstanding output from the card, notifying you if any data was lost.

**Note** This command affects *only* the node on which the CLI is running when you execute it, regardless of a target set with the command **set snmp hostname** *name*.

## Syntax

**connect** *card#* **[force] [diagnostic]**

## Arguments

**Note** The **force** and **diagnostic** arguments are optional.

- *card#*

  The card number, as follows:

  1 — NP (if there is an NP in slot 2, it must be active)

  2 — NP (NP in slot 1 must be active) or line card (populate slots 3–10 before slot 2)

  3–10 — Line cards

- **force**

  Forces this CLI session to take control of the slot even if the slot was being used by another CLI in the network.

- **diagnostic**

  Used only when connecting to a card to run diagnostics. See the *LightStream 2020 Installation and Troubleshooting Manual*.

# loadcard

Load a file into a card, start it, and establish a console connection with its TCS slave. Requires protected mode.

## Description

The **loadcard** command loads a line card program, such as the operational software or a diagnostic program, into the specified card. The command resets the card, then loads and starts the line card software.

**Note** This command affects *only* the node on which the CLI is running when you execute it, regardless of a target set with the command **set snmp hostname** *name*.

## Syntax

**loadcard** *card#* **[***load-address***] [***filename***]**

## Arguments

**Note** The *load-address* and *filename* arguments are optional.

- *card#*

  The card number, as follows:

  1 — NP (if there is an NP in slot 2, it must be active)

  2 — NP (NP in slot 1 must be active) or line card (populate slots 3–10 before slot 2)

  3–10 — Line cards

- *load-address*

  The load address. Normally, you omit this argument and use the default load address.

- *filename*

  The name of the file containing the program you want to load. If no filename is specified, this command loads the operational software for the card.

# test

Determine whether or not a specified card is functional.

## Description

The **test** command runs diagnostic tests on a specified card to determine whether it should be replaced. The software automatically identifies the type of card and runs appropriate tests. (PLC/FDDI, PLC/Ethernet, and CLC/OC-3c modules are not tested.) You must specify the card number (3–10).

If a card fails, record the displayed error codes and report them when returning the card.

**Note** You cannot run diagnostics from the CLI on an NP. To test an NP, you must deactivate the node, load diagnostics manually, and run tests through the manufacturing interface. See the *LightStream 2020 Installation and Troubleshooting Manual* for information about this procedure, and for information about replacing a defective card.

**Note** This command affects *only* the node on which the CLI is running when you execute it, regardless of a target set with the command **set snmp hostname** *name*.

## Syntax

**test** <*card#*> **[-l][-p][-s][-x][-F***file***]**

**test** <*card#*> **-r**

**test** <*card#*> **-m [-F***file***]**

## Switches

The default behavior, with only the *card#* argument, is to run standard tests in the background (see the -r switch).

**-F***file*

Load diagnostics from *file* rather than from the default file for the card. *file* can only be a copy of the default file, and you must ensure that *file* matches the card type. In the absence of this switch, the default files are automatically matched with the card type and loaded. They are as follows:

- Low Speed Card — diag_ls1.aout
- Medium Speed Card — diag_ms1.aout

**-l**

If tests require looping plugs or cables (rare), use this switch.

**-m**

Stay connected and monitor output of standard PASS/FAIL tests plus manufacturing tests giving more detailed data until completion or timeout. A test in progress may be terminated with [**^C**] These tests are equivalent to using **loadcard** with the manufacturing diagnostics and connecting to the card (see the *LightStream 2020 Installation and Troubleshooting Manual*).

---

**Note** The -r switch cannot be used to retrieve output of **test -m**.

---

**-p**

Stay connected and poll standard test output approximately every second until completion or timeout. Tests completed between these snapshots are not displayed, and if a test runs longer than one polling interval, successive dots indicate the successive polls. The -r switch may be used after completion. A test in progress may be terminated with [**^C**].

**-r**

Retrieve and display status of standard diagnostics previously run in the background. The **test** command changes card status to test, and the -r switch may be used until the card status changes to active or inactive or until the card is loaded with some other software, overwriting the memory locations in which test results are stored. A heartbeat indication in the display serves to confirm that the diagnostics are running in case the same test is active over successive polls with **test -r**. Base your estimate of test run time on experience with **test -p**.

---

**Note** The -r switch cannot be used to retrieve results of manufacturing diagnostics (obtained with the -m switch).

---

**-x**

Extended test, running long memory tests.

# read

Read memory and hardware registers accessible by the TCS.

### Description

This command is only for use by LightStream support personnel. It is of no value without detailed knowledge of memory locations, hardware registers, their contents, and their functions in the LightStream node.

## write

Write memory and hardware registers accessible by the TCS. Requires protected mode.

**Caution** This command is only for use by LightStream support personnel. Without detailed knowledge of memory locations, hardware registers, their contents, and their purposes, use of this command will almost certainly destroy the functioning of the LightStream node.

**3 x 3 x**

# Setting and Displaying Configuration Attributes

This chapter lists the parameters that you can configure with the LightStream configuration tool on a workstation, and shows the functionally equivalent CLI commands for displaying and setting a node's configuration if the configuration tool is not available to you.

- The first part of this chapter, "Setting Configuration Attributes," tells how to set the values of configuration attributes with CLI **set** and **setsnmp** commands. Various tables show what commands to use for each configuration attribute.
- The second part of this chapter, "Displaying Configuration Parameters," contains tables that show how to display the current values of configuration parameters with CLI **show** and **getsnmp** commands.

---

**Note** The MIB object identifier *portID* is a port identifier, as described in the section "Port Identifiers." For example, the MIB object that stores the alias or name for port 3 on card 4 is portInfoName.3004 unless it is an Ethernet port (3104) or an FDDI port (3204).

---

## Setting Configuration Attributes

This section describes how to change configuration parameters with CLI commands, in case the configuration tool is not available.

### Where Configuration Information is Stored

Configuration information is stored in five locations:

**1** Pre-programmed defaults.

2 Run-time memory on the target node.

3 A copy in on-board EEPROM of the run-time values of certain line card attributes.

4 The local configuration database on the disk of the target node.

5 The global configuration database on the network management station (NMS) on which the configuration tool is running.

## How to Change Attribute Values

The preferred method of setting attribute values in the local configuration database is by using the configuration tool on the network management station (NMS). See the *LightStream 2020 Configuration Guide* for details.

CLI commands may be used to change the values of configuration attributes. Normally, configuration changes made with CLI commands are made to run-time memory only. The next time the node is reset, these volatile changes are overwritten. This is useful for testing proposed configuration changes.

CLI commands may also be used to change the local configuration database. CLI set commands write to the local database as well as to run-time memory after you enter the following command:

```
*cli> set config lock
```

The **set config lock** command also locks the MMA so that a CLI user connected from a different IP address cannot concurrently make configuration changes. The CLI issues a periodic reminder that the MMA is locked.

Use the following command to restore the default state (unlocked, no writes to database):

```
*cli> set config unlock
```

These commands are equivalent to setting the mmaSetLock MIB object to 3 (locked) or to 1 (unlocked). There is a third value of the mmaSetLock object that cannot be set by the **set config** command. If you are testing configuration changes, but do not want to save them yet in the local configuration database, you can use the following command to lock the MMA (so that a user connected from a different IP address cannot concurrently make configuration changes) *without* causing CLI sets to write to the local database:

```
*cli> setsnmp mmaSetLock.0 2
```

You can restore the default state with the **set config unlock** command. The lock is automatically removed after two minutes of inactivity or when the CLI session is closed.

For additional details about these commands, refer to the description of the **set config** command.

## Synchronizing Databases

After you make changes to the local configuration database with CLI commands, as described above, it is out of synch with the global database. The global configuration database is located on the NMS on which the configuration tool is running. To copy local configuration changes to the global database, you must use the verify function in the configuration tool. This function retrieves the local settings and allows you to write them over the global values. See the *LightStream 2020 Configuration Guide*.

## Working with Temporary Configuration Changes

CLI changes in run-time memory may cause confusion.

### How Attributes are Re-Initialized

When the NP is restarted (e.g. when the node is rebooted), or, for card or port attributes, when a card comes up, attribute values in run-time memory are reset in the following sequence:

- Pre-programmed defaults are read in to run-time memory.
- Run-time values of some port attributes are read in from on-board EEPROM and overwrite the default settings.
- Values stored in the local configuration database overwrite any prior values.

---

**Note** The result can be a combination of database settings, settings stored in EEPROM (which might have been mistakenly thought to be temporary changes in run-time memory), and defaults.

---

### Port Attributes that are Stored in EEPROM

Run-time values of the following port attributes are stored in EEPROM on the line card:

| Line Card Attribute | Card Type |
|---|---|
| edge/trunk mode | LSC, MSC, CLC |
| cell payload scrambling | MSC, CLC |
| clocking (internal/external) | CLC |
| bitrate | LSC |
| DCE/DTE mode | LSC |
| cable length | MSC |
| C-bit Parity/Clear Channel | MSC |
| HEC/PLCP mode | MSC |

Normally, when you set the status of a card to inactive and then to active, changes made with CLI commands to run-time memory (but not to the database) are lost. However, changes to the above attributes are read back into run-time memory from EEPROM when the card comes up.

A value read in from EEPROM remains in effect if the default value for that attribute was never changed in the local configuration database.

For example, suppose you have used the CLI **set port** *c.p* **characteristics** command to specify the DCE bit rate for a port. This attribute is not configured in the local database because you have been using the default bit rate for this port, or perhaps you are configuring a new card with CLI commands. Because this was an experimental setting, you did not first use the **set config lock** command to write this change to the local database. The new run-time value of this attribute is stored in EEPROM on the line card. When the card is disabled and enabled (or the NP restarted), NP software reads EEPROM and writes the bit rate to run-time memory. There it remains because there is no bit rate setting in the local database to overwrite it.

### Node Attributes that Look Like Port Attributes

A similar confusion is due to thinking of certain node attributes mistakenly as edge card and port attributes. These are the attributes concerned with the following LAN-related functions:

- Workgroups
- Filters
- Spanning tree bridge
- Static bridge routes
- The default bridging action for a port

These attribute values are stored in the NP's run-time memory and not on the card. Consequently, they are overwritten from the local configuration database only when the NP is restarted, typically when the node is rebooted. They are not overwritten when a card is disabled and re-enabled, or even when the card is physically removed and reinstalled.

For example, suppose you have used the CLI to specify some custom filters and assign them to a port, but you did not first use the **set config lock** command to write them to the local database. These filter settings are retained in NP memory. When the edge card is disabled and re-enabled, the filter settings are not affected. You should use CLI commands or the configurator to delete them. They are also removed when you reboot the node or in some other way restart the NP.

## Chassis and Switch Attributes

Table 3-1 shows the configurable chassis attributes and the corresponding CLI **set** and **setsnmp** commands:

**Table 3-1 Chassis Attributes and CLI set and setsnmp Commands**

| | |
|---|---|
| **System Attributes** | |
| Chassis ID | **setsnmp chassisId.0** *ID#* |
| Name | **set chassis name** *name* |
| Contact | **setsnmp sysContact.0** *"Contact information"* |
| Location | **setsnmp sysLocation.0** *"Location information"* |
| **IP Address Attributes** | |
| Primary IP Address | **set chassis activeip** *IPaddress* |
| Default Router | **set chassis defrouter** *IPaddress* |
| NP IP Address | **set chassis ethernetaddr** *IPaddress* |
| NP IP Mask | **set chassis ethernetmask** *mask* |
| Secondary IP Address | **set chassis secondaryip** *IPaddress* |
| Subnet Mask | **set chassis netmask** *mask* |
| **SNMP Agent Attributes** | |
| Trap Filter | **set cli traplevel {oper\|info\|trace\|debug}** |
| Trap Log Status | **set chassis traplog {on\|off}** |

The following switch-wide attribute may be configured only in expert mode in the configuration tool.

| Card Attributes | |
|---|---|
| Max VCs for This Card | **setsnmp cardMaxVCs.***card# nnn* |

## Card Attributes

Following are the configurable card attributes and the corresponding CLI **set** and **setsnmp** commands.

| Card Attributes | |
|---|---|
| Admin Status | **set card {active\|inactive\|testing}** |
| Name | **setsnmp cardName.***card# name* |
| Type | **setsnmp cardBoardType.***card# type* |

## Port Attributes

Following are the configurable port attributes and the corresponding CLI **set** and **setsnmp** commands. There are common port attributes, low-speed trunk and edge port (LSC) attributes, T3 and E3 medium-speed trunk and edge port (MSC) attributes, FDDI port attributes, and OC3 trunk and edge port attributes. (For Ethernet ports, only the two attributes common to all ports may be configured.)

| Common Port Attributes | |
|---|---|
| Port Name | **setsnmp portInfoName.***portID name* |
| Port Status | **set port** *c.p* **{active\|inactive\|testing}** |
| **LSC Port Attributes** | |
| DCE Bit Rate | **set port** *c.p* **characteristics dce-bitrate** *Kbytes* |
| DCE/DTE Type | **set port** *c.p* **characteristics dce-dte-type {dce\|dte\|dce-internal}** |
| DTE Bit Rate | **set port** *c.p* **characteristics dte-bitrate** *Kbytes* |
| Net Error Threshold | **setsnmp frProvMiNetErrorThreshold.***portID n* |
| Full Enquiry Interval | **setsnmp frProvMiUserFullEnquiryInterval**.*portID n* |
| LMI Type | **set port** *c.p* **lmiconfig {none\|frif\|ansi_t1_617d\|q933a}** |
| Max Frame Size | **setsnmp edgeMaxFrameSize**.*portID bytes* |
| Max Supported VCs | **setsnmp frProvMiMaxSupportedVCs.***portID n* |
| Net Monitored Events | **setsnmp frProvMiNetMonitoredEvents.***portID n* |
| Net Interface Type | **set port** *c.p* **netinterfacetype {uni\|nni}** |
| Net Request Interval | **setsnmp frProvMiNetRequestInterval.***portID sec* |
| Polling Interval | **setsnmp frProvMiUserPollingInterval.***portID sec* |
| User Error Threshold | **setsnmp frProvMiUserErrorThreshold.***portID n* |
| **T3 and E3 MSC Port Attributes** | |
| Cable Length | **setsnmp ms1InfoAdminCableLength.***portID n* |
| DS3 Line Type (T3 only) | **setsnmp dsx3LineType.***portID* **{4\|5\|6\|8}** (default = 5 if T3, 6 if E3/G.804, 8 if E3/Scrambling) |
| Cell Payload Scrambling | **setsnmp ms1InfoAdminScramble.***portID* **{1\|2}**<br>**setsnmp clc1InfoAdminScramble.***portID* **{1\|2}** |

| FDDI Port Attributes | |
|---|---|
| Link Error Rate Alarm | **setsnmp fddimibPORTLerAlarm.***SMT.SMTport n* |
| Link Error Rate Cutoff | **set port c.p fddi {aport\|bport} lercutoff** *rate* |
| Notify Timer | **setsnmp fddimibSMTTNotify.***fddimibSMTIndex* |
| **OC3 Port Attributes** | |
| Clocking Type | **setsnmp clc1InfoAdminClock.***portID* |
| Cell Payload Scrambling | **setsnmp clc1InfoAdminScramble.***portID* |

## Internetworking Attributes

Following are the configurable attributes and the corresponding CLI **set** commands for internetworking.

| Spanning Tree Attributes | |
|---|---|
| Forward Delay | **set stb forwdelay** *t* |
| Hello Time | **set stb hellotimer** *t* |
| Max Age | **set stb maxage** *age* |
| Priority | **set stb priority** *pri* |
| Path Cost | **set port** *c.p* **stb pathcost** *n* |
| Port Priority | **set port** *c.p* **stb priority** *n* |
| STB Enabled | **set port** *c.p* **stb {enable\|disable}** |
| **Custom Filters Attributes** | |
| B'cast Limit | **set port** *c.p* **bcast-limit fps** |
| Card | (The *c* in syntax of **set port** *c.p* commands) |
| Port Default Action | **set port** *c.p* **bflt-def** *ID* **{block\|forward}** |
| Filter ID | (The *ID* in **define bflt** *ID filter-exp*) |
| Filter Expression | **define bflt** *ID filter-expr* |
| Forward or Block | **set port** *c.p* **bflt** *ID* **{block n\|forward n}** |
| ID | **set port** *c.p* **bflt** *ID* **{block n\|forward n\|delete}** |
| Port | (The *p* in syntax of **set port** *c.p* commands) |
| Priority | (The *p* in **set port** *c.p* **bflt** *action n*) |
| **Static Route Attributes** | |
| MAC | **set stb static** *MACaddr* **rcv {***c.p***\|any} xmit** *c.p* |
| Receive Port | (The **rcv** argument in a **set stb static** command) |
| Transmit Ports | (The **xmit** argument in a **set stb static** command) |
| **SNMP Options** | |
| Trap Address | **set snmp hostname {***name***\|***IPaddress***}** |
| Trap Community Name | **set snmp community** *name* |

## PVC Attributes

Following are the configurable attributes and the corresponding CLI **set** and **setsnmp** commands used to provision PVCs. The source node is referred to as node A, and the destination node as node B.

The CLI commands take the source node, port, and VCI number from the target host. You must set the PVC attributes from both ends and then activate the PVC as follows:

**set port** *c.p* **{dlci|frameforwarding|vci} activate**

| **Source (Node A) Attributes** | |
|---|---|
| DLCI A | (The *dlci#* in **set port** *c.p* **dlci** *dlci#* commands) |
| A Insured Rate | **set port** *c.p* **dlci** *dlci#* **insured-rate** *bps*<br>**set port** *c.p* **frameforwarding insured-rate** *bps*<br>**set port** *c.p* **vci** *vci#* **insured-rate** *cps* |
| A Max. Rate | **set port** *c.p* **dlci** *dlci#* **max-rate** *bps*<br>**set port** *c.p* **frameforwarding max-rate** *bps*<br>**set port** *c.p* **vci** *vci#* **max-rate** *cps* |
| A Node | (Same as B Node, but while connected to other host) |
| A Port | (Same as B Port, but while connected to other host) |
| VCI A | (Same as B VCI, but while connected to other host) |
| **Destination (Node B) Attributes** | |
| DLCI B | **set port** *c.p* **dlci** *A-dlci#* **destdlci** *B-dlci#* |
| B Insured Rate | **set port** *c.p* **dlci** *dlci#* **insured-rate** *bps*<br>**set port** *c.p* **frameforwarding insured-rate** *bps*<br>**set port** *c.p* **vci** *vci#* **insured-rate** *cps* |
| B Max. Rate | **set port** *c.p* **dlci** *dlci#* **max-rate** *bps*<br>**set port** *c.p* **frameforwarding max-rate** *bps*<br>**set port** *c.p* **vci** *vci#* **max-rate** *cps* |
| B Node | **set port** *c.p* **dlci** *dlci#* **destnode {***S/N\|IPaddr***}**<br>**set port** *c.p* **frameforwarding destnode {***S/N\|IPaddr***}**<br>**set port** *c.p* **vci** *vci#* **destnode {***S/N\|IPaddr***}** |
| B Port | **set port** *c.p* **dlci** *dlci#* **destport** *c.p*<br>**set port** *c.p* **frameforwarding destport** *c.p*<br>**set port** *c.p* **vci** *vci#* **destport** *c.p* |
| B VCI | **set port** *c.p* **vci** *vci#* **destvci** *destvci#* |

Following are the configurable per-port call setup and bandwidth attributes and the corresponding CLI **set** and **setsnmp** commands. These PVC attributes may be configured only in expert mode.

| | |
|---|---|
| B Insured Burst | Same as Source Insured Burst Size, from destination host |
| B Max. Burst | Same as Source Maximum Burst Size, from destination host |
| A Insured Burst | **set port** *c.p* **dlci** *dlci#* **insured-burst** *bps*<br>**set port** *c.p* **frameforwarding insured-burst** *bps*<br>**set port** *c.p* **vci** *vci#* **insured-burst** *cps* |
| A Max. Burst | **set port** *c.p* **dlci** *dlci#* **max-burst** *bps*<br>**set port** *c.p* **frameforwarding max-burst** *bps*<br>**set port** *c.p* **vci** *vci#* **max-burst** *cps* |

| | |
|---|---|
| Principal Service Type | **set port** *c.p* **vci bwtype** |
| Transmit Priority | **set port** *c.p* **vci priority** |
| Secondary Scale | **setsnmp edgeSecondaryScale.***portID n* |

## VLI Attributes

To add or delete a workgroup, use one of the following commands:

cli> **set port** *c.p* **wgrp add** *ID*

cli> **set port** *c.p* **wgrp delete** *ID*

To change the sense of a workgroup list for a port, use one of the following commands:

cli> **set port** *c.p* **wgrp include** *ID*

cli> **set port** *c.p* **wgrp exclude** *ID*

In practically every case, the workgroup list for a port should be configured in the **include** sense, to allow intercommunication with all ports that are members of the listed workgroups. There is one list per port with up to seven IDs in the **include** sense, up to six in the **exclude** sense (all 65,534 groups *except* those listed).

# Displaying Configuration Parameters

The following sections contain tables that show how to display the current values of configuration parameters with CLI **show** commands.

## Chassis and Switch Attributes

Following are the configurable chassis attributes and the corresponding CLI **show** commands:

| **System Attributes** | |
|---|---|
| Chassis ID | **show chassis general** |
| Name | |
| Contact | |
| Location | |
| **IP Address Attributes** | |
| Primary IP Address | **show chassis general** |
| Default Router | |
| NP IP Address | |
| NP IP Mask | |
| Secondary IP Address | |
| Subnet Mask | |
| **SNMP Agent Attributes** | |
| Trap Filter | **show chassis agent** |
| Trap Log Status | |

## Card Attributes

Following are the configurable card attributes and the corresponding CLI **show** commands.

| Card Attributes | |
|---|---|
| Admin Status | **show card** *n* **status** |
| Name | **show card** *n* **name** |
| Type | **show card** *n* **hardware** |

One card attribute can be set only in expert mode with the configuration tool, and is displayed with a **getsnmp** command:

| Card Attributes | |
|---|---|
| Max VCs for This Card | **getsnmp cardMaxVCs.***card#* |

## Port Attributes

Following are the configurable port attributes and the corresponding CLI **show** and **getsnmp** commands. There are common port attributes, low-speed trunk and edge port (LSC) attributes, T3 and E3 medium-speed trunk and edge port (MSC) attributes, FDDI port attributes, and OC3 trunk and edge port attributes. (For Ethernet ports, only the two attributes common to all ports may be configured.)

| Common Port Attributes | |
|---|---|
| Port Name | **show port** *c.p* **name** |
| Port Status | **show port** *c.p* **status** |
| **LSC Port Attributes** | |
| DCE Bit Rate | **show port** *c.p* **physical** |
| DCE/DTE Type (Admin Net Interface Type) | |
| DTE Bit Rate | |
| Max Frame Size | |
| Net Error Threshold | **show port** *c.p* **framerelay** |
| Full Enquiry Interval | |
| LMI Type | |
| Max Supported VCs | |
| Net Monitored Events | |
| Net Interface Type | |
| Net Request Interval | |
| Polling Interval | |
| User Error Threshold | |
| **T3 and E3 MSC Port Attributes** | |
| Cable Length | **show port** *c.p* **physical** |
| DS3 Line Type (T3 only) | |
| Cell Payload Scrambling | |
| **FDDI Port Attributes** | |

| | |
|---|---|
| Link Error Rate Alarm | **show port** *c.p* **fddi port{aport\|bport}** |
| Link Error Rate Cutoff | |
| Notify Timer | **getsnmp fddimibSMTTNotify.***fddimibSMTIndex* |
| **OC3 Port Attributes** | |
| Clocking Type | **getsnmp clc1InfoAdminClock.***portID* |
| Cell Payload Scrambling | **show port** *c.p* **physical** |

## Internetworking Attributes

Following are the configurable attributes and the corresponding CLI **show** commands for internetworking.

| | |
|---|---|
| **Spanning Tree Attributes** | |
| Max Age | **show stb general** |
| Hello Time | |
| Forward Delay | |
| Priority | |
| Path Cost | **getsnmp dot1dStpPortPathCost.***portID* |
| Port Priority | **getsnmp dot1dStpPortPriority.***portID* |
| STB Enabled | **getsnmp dot1dStpPortEnable.***portID* |
| **Custom Filters Attributes** | |
| Filter ID | **show port** *c.p* **bflt** |
| Port Default Action | |
| Filter Expression | **show bflt ID** |
| Forward or Block | |
| Priority | |
| Card | — |
| Port | — |
| B'cast Limit | **show port** *c.p* **bcast-limit** |
| **Static Route Attributes** | |
| MAC | **show stb static** |
| Receive Port | |
| Transmit Ports | |
| **SNMP Options** | |
| Trap Address | **show snmp** |
| Trap Community Name | |

## PVC Attributes

Following are the configurable attributes and the corresponding CLI **show** commands used to provision PVCs. The source node is referred to as chassis A, and the destination node as chassis B.

The CLI commands take the source node, port, and VCI number from the target host. You may display the corresponding attributes after changing the target host as follows:

**setsnmp hostname** *name*

| Source (Node A) and Destination (Node B) Attributes | |
|---|---|
| DLCI A (B) | **show port** *c.p* **listdlci** |
| | **show port** *c.p* **frameforeward** |
| A (B) Insured Rate | **show port** *c.p* **dlci** *dlci#* |
| A (B) Max. Rate | **show port** *c.p* **frameforward** |
| | **show port** *c.p* **vci** *vci#* |
| A (B) Node | **show chassis listdlci** |
| A (B) Card | **show port** *c.p* **listdlci** |
| A (B) Port | **show chassis listvci** |
| | **show port** *c.p* **listvci** |
| VCI A (B) | **show chassis listvci** |
| | **show port** *c.p* **listvci** |

The following attributes may only be set in Expert Mode in the configuration tool:

| Source (Node A) and Destination (Node B) Attributes | |
|---|---|
| A (B) Insured Burst | **show port** *c.p* **dlci** *dlci#* |
| A (B) Max. Burst | **show port** *c.p* **frameforward** |
| | **show port** *c.p* **vci** *vci#* |
| Principal Service Type | **getsnmp frCktAdminPrinBwType.***portID.dlci#* |
| | **getsnmp ffCktAdminPrinBwType.***portID.dlci#* |
| | **getsnmp sUniCktAdminPrinBwType.***portID.dlci#* |
| | **getsnmp pvcCktAdminPrinBwType.***portID.dlci#* |
| Transmit Priority | **getsnmp frCktAdminTransPri.***portID.dlci#* |
| | **getsnmp ffCktAdminTransPri.***portID.dlci#* |
| | **getsnmp sUniCktAdminTransPri.***portID.dlci#* |
| | **getsnmp pvcCktAdminTransPri.***portID.dlci#* |
| Secondary Scale | **getsnmp edgeSecondaryScale.***portID* |

The attributes for the destination (Node B) are identical to those for the source (Node A). To display them you must first set the target to Node B with the **set snmp hostname** *name* command.

## VLI Attributes

To display VLI attributes, use the **show port** *c.p* **wgrp** command.

Network management systems that use SNMP work with information stored in a Management Information Base (MIB). This chapter:

- Describes the MIB used to manage LightStream 2020 enterprise ATM switches.
- Tells how to use this information to identify MIB objects by their addresses in the MIB. The SNMP commands (getsnmp, getnextsnmp, setsnmp, and walksnmp) and the browse command require MIB addresses as arguments. These commands are described in the "CLI Command Reference" chapter.

---

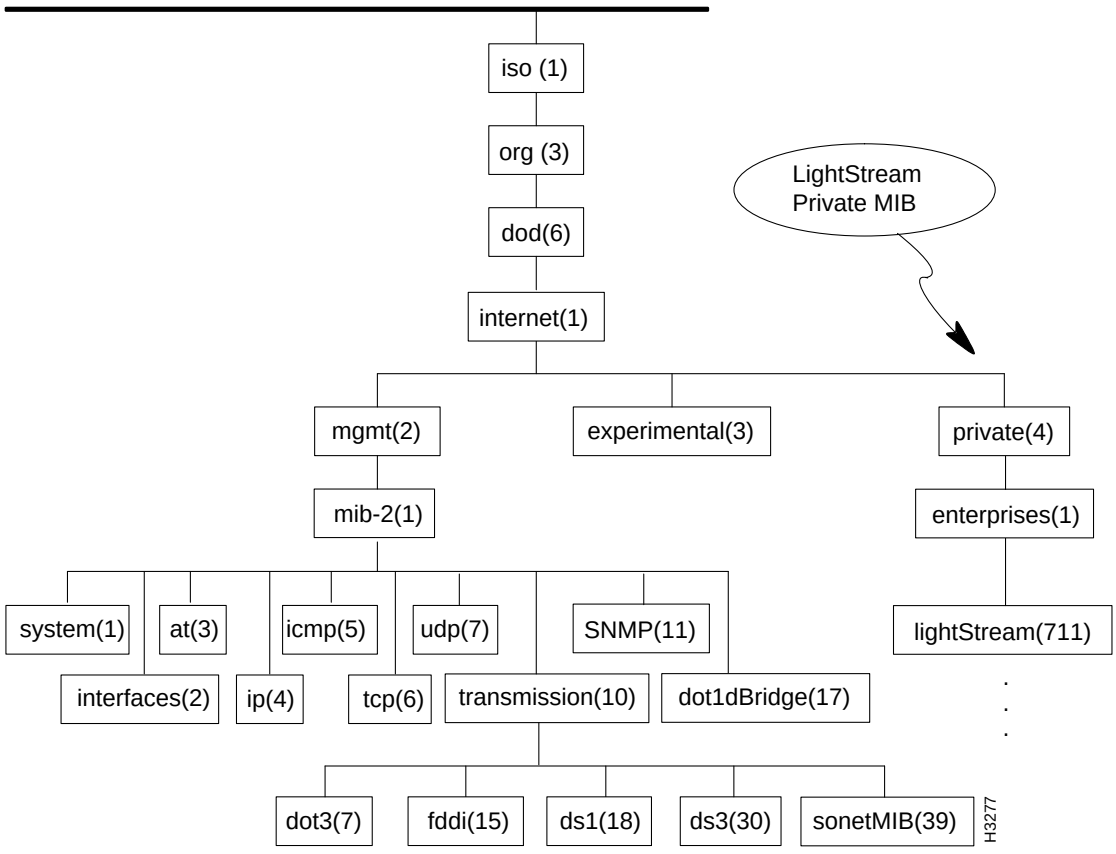
**Note** MIB objects whose values may be read but not written are marked *RO* (read only) in the tables of this chapter. Other MIB objects, unmarked in the rightmost columns of the tables, may be both read and written. (The table and table entry objects, like cardTable and cardEntry, are not accessible. They are shown here only because they help to structure the MIB.)

---

# MIB Overview

Figure 4-1 shows a high-level diagram of the MIB that is used to manage LightStream switches. The organization of the MIB is defined in a document called the *Structure of Management Information* (SMI).

**Figure 4-1 MIB Tree Structure**

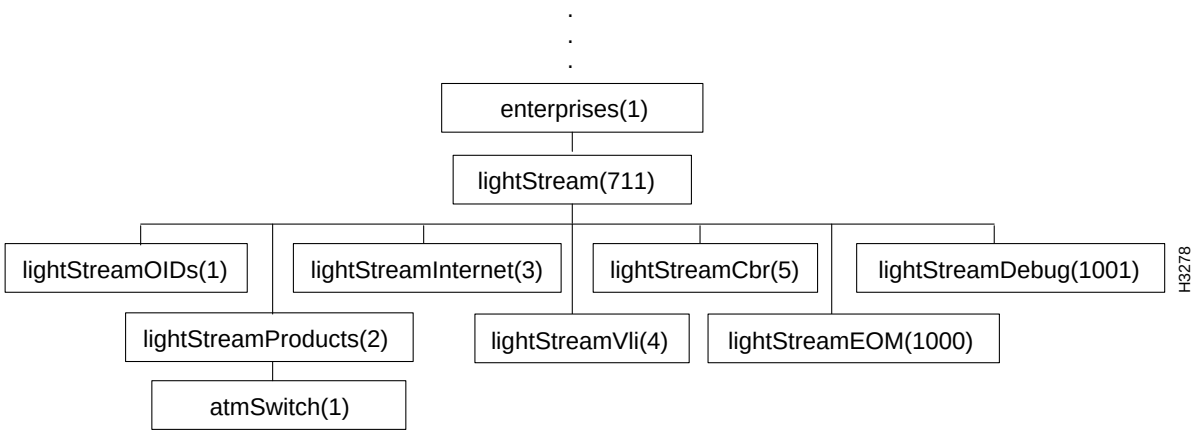


For information on some of the standard MIBs shown in Figure 4-1, refer to the following documents:

- MIB-II (RFC 1213)
- Dot-3 Ethernet (RFC 1398)
- FDDI (RFC 1512)
- RMON MIB (RFC 1271)
- DS1 MIB (RFC 1232)
- DS3 MIB (RFC 1233)
- Dot-1 Bridge (RFC 1493)
- SONET (RFC 1595)

Under the Private subtree is the LightStream private MIB that is described in the remainder of this chapter. The high level structure of the LightStream private MIB is shown in Figure 4-2:

**Figure 4-2 LightStream Private MIB Tree Structure**




**Note** The ellipsis in Figure 4-2 indicates where the upper levels of the MIB (the sequence of MIB objects described by the path iso.org.dod.internet.private) are not shown; see Figure 4-1 for that information.

For detailed descriptions of those MIB objects that can be displayed and set, see the figures and tables in the following sections:

- The atmSwitch Subtree
- The lightStreamInternet Subtree
- The lightStreamVLI Subtree

**Note** The lightStreamOIDs branch appears in Figure 4-2 and is displayed by the **browse** command, but is not described here because the objects in it are inaccessible (they cannot be displayed or set).

Examine the MIB files themselves for information at a level of detail much greater than can be given here.

# MIB Addresses

A MIB address identifies a particular MIB object by its location in the MIB. Five CLI commands require MIB addresses in their arguments: the **browse** command, and the SNMP commands (**getsnmp**, **getnextsnmp**, **setsnmp**, and **walksnmp**). These commands are for monitoring and controlling the LightStream switch, as described in the "CLI Command Reference" chapter, in ways that are not available with the standard CLI control commands (see Table 2-2 for a list).

## Alternative Forms of MIB Addresses

You can specify the MIB address in a number of ways:

- A variable name: sysContact
- A "path" of dot-separated numbers (the numbers in parentheses in Figure 4-1 and Figure 4-2): 1.3.6.1.2.1.1.4
- A combination, where the first element is a variable name and the numbers specify one branch in the subtree under it: mgmt.1.1.4

Thus, the following addresses all refer to the same MIB variable:

- 1.3.6.1.2.1.1.4
- mgmt.1.1.4
- mib.1.4
- system.4
- sysContact

## Determining MIB Variable Names

To learn the names of the MIB variables in one of the subtrees shown in Figure 4-1 or Figure 4-2, you can specify the name of the subtree as the argument of the **walksnmp** command. This command displays all the "leaves" below the specified point in the MIB. You may also use the **browse** command to step through the branches of the MIB tree and, when you reach the end of a branch, to display the values of leaves one at a time.

## Object Identifiers

The **getsnmp**, **getnextsnmp**, and **setsnmp** commands display and change the value of a single "leaf," that is, a single instance of a given MIB object. To identify a single instance of a MIB object at the end of a branch, you specify the object name followed by a numeric suffix known as an object identifier.

If there is only a single instance of the given object, then the identifier suffix for that instance is 0. For example, there is only one instance of the sysContact object, so its identifier is sysContact.0. The following command displays the name of the contact person for the target node:

cli> **getsnmp sysContact.0**

Just as with the SNMP commands and the **browse** command, described above, the first part of the object address, the MIB variable, may be specified in a number of ways. Thus, the following addresses all refer to the same MIB object:

- 1.3.6.1.2.1.1.4.0
- mgmt.1.1.4.0
- mib.1.4.0
- system.4.0
- sysContact.0

This is the simplest form of object identifier. More complex types include port identifiers and object identifiers for multiply-indexed objects.

## Port Identifiers

The usual way of specifying a port number in the argument of a CLI command is in the format *c.p*, where *c* is the card (slot) number and *p* is the number of the port on that card. However, the **show** command usually displays the value of the ifIndex object for the interface, which gives the port number in a more complex format. For example, the following command displays the ports on card 4:

```
cli> show card 4 ports
Port 4000 Frame Relay Name: larch.4.0
Port 4001 Frame Relay Name: larch.4.1
Port 4002 Frame Relay Name: larch.4.2
cli>
```

when a MIB object identified by port number is given as the argument of the **getsnmp**, **getnextsnmp**, or **setsnmp** command, this more complex format is used. For example, the following command tells us which data link connection management scheme is active on frame relay port 4.2 (card 4, port 2), specified here as port 4002:

```
cli> getsnmp frProvMiState.4002
Name: frProvMiState.4002 Value: 3
cli>
```

By looking this object up in Table 4-13, or by searching for it in the MIB file private_mib.asn, we find that the value 3 means that the default ANSI standard T1.617 Annex D is active on this port.

In effect, the long form of port number is *ctpp* or *cctpp*, where *c* is the card number, *p* is the port number, and *t* is the card type (2=FDDI, 1=Ether, 0=Other). The formula for determining the port number in this long format is as follows:

((*c* * 1000) + *p* + *t*)

The variables in this formula are as follows:

- *c* — The card number, an integer in the range 1-11. A number in the range 1-10 is the number of the physical slot in which the card resides. The number 11 represents the logical interface for the LightStream ATM network.
- *p* — The port number on the card, an integer in the range 0–7. If *c* is 11 (the logical interface for the LightStream ATM network) then *p* must be 4.
- *t* — An offset indicating the card type. This is an arbitrary numeric value assigned to each type of card. In Release 2.0, the values of *k* are as follows:

  100 — Ethernet

200 — FDDI

0 — Other

For example, the MIB object that stores the alias or name for port 3 on card 4 is portInfoName.3004, or portInfoName.3104 if it is an Ethernet port, or portInfoName.3204 if it is an FDDI port.

**Note** Port number 1255 or 2255 refers to the control port (port 255) of the NP in slot 1 or slot 2, respectively.

## Multiply-Indexed Objects

Some objects require us to specify more than one index number. For example, the following command tells us that the local LMI state of the frame relay circuit on DLCI 16, port 4, card 2 is inactive:

```
cli> getsnmp frCktInfoLclLMI.4002.16
Name: frCktInfoLclLMI.4002.16 Value: 2
cli>
```

The double index 4002.16 means that this frCktInfoLclLMI object is indexed first by port number 4002 (equivalent to port 4.2) and then by DLCI number 16. For the fact that the value 2 means "inactive", see Table 4-15, or see the description text for frCktInfoLclLMI in the private_mib.asn MIB file.

In LightStream Release 2.0, the doubly-indexed entries in the LightStream private MIB include the following:

| Table Entry Object | Indexing Objects |
|---|---|
| frCktEntry | frCktSrcIfIndex, frCktSrcDlci |
| frCktInfoEntry | frCktInfoIfIndex, frCktInfoDlci |
| sUniCktEntry | sUniCktSrcIfIndex, sUniCktSrcVCI |
| sUniCktInfoEntry | sUniCktInfoIfIndex, sUniCktInfoVCI |
| pvcEntry | pvcSrcIfIndex, pvcSrcPvcId |
| lsFrameRelayDlciStatEntry | frameRelayDlciStatPortIndex, frameRelayDlciStatDlciIndex |
| lsEdgePortToSwMsgLenEntry | edgeToSwMsgLenPortIndex, edgeToSwMsgLenBinIndex |
| lsEdgeSwToPortMsgLenEntry | edgeToPortMsgLenPortIndex, edgeToPortMsgLenBinIndex |
| lsEdgeCpuWorkloadEntry | lsEdgeWorkloadCardIndex, lsEdgeWorkloadTypeIndex |
| lsFrameForwardStatEntry | frameForwardStatPortIndex, frameForwardStatIndex |
| lsTrunkCpuWorkloadEntry | lsTrunkWorkloadCardIndex, lsTrunkWorkloadTypeIndex |
| fsuPortXmtCellsEntry | fsuXmtCellsPortIndex, fsuXmtCellsPriorityIndex |
| fsuQueueCellLenEntry | fsuQueueCellLenPortIndex, fsuQueueCellLenSubQIndex |
| fsuDropEventEntry | fsuDropEventPortIndex, fsuDropEventWatermarkIndex |
| lsCellVciStatEntry | cellVciStatPortIndex, cellVciStatVciIndex |
| gidLineCardEntry | gidLineCardChassis, gidLineCardSlot |
| gidPortEntry | gidPortChassis, gidPortID |

| Table Entry Object | Indexing Objects |
|---|---|
| lightStreamBridgeFilterEntry | lightStreamBrFilterId, lightStreamBrFilterTokenIndex |
| lightStreamBridgeFilterParameterEntry | lightStreamBrFilterParmPort, lightStreamBrFilterParmFilterId |
| lightStreamVliPortWorkGroupEntry | lightStreamVliPortWorkGroupPort, lightStreamVliPortWorkGroupID |

Objects in standard MIBs may also be multiply indexed. These appear with more than one number in the object identifier when you walk or browse the MIB. You can also identify these objects by examining the MIB file in which they are defined: the INDEX entry of the definition lists more than one object name as indexing objects.

An example is the fddimibPATHEntry object defined in the fddi.asn MIB file. The FDDI stations implemented in the LightStream node are single-MAC, dual-attachment FDDI stations. There are two paths per SMT. (An FDDI path is the sequence of MACs and ports in token order.) The paths are managed by the fddimibPATHTable object. The fddimibPATHTable object is defined as a sequence of fddimibPATHEntry objects.

The fddimibPATHEntry object is doubly indexed. The first indexing object is fddimibPATHSMTIndex, and the second is fddimibPATHIndex. LightStream software calculates the values of these SMT index objects as follows:

- There are two SMTs per card, indexed as 2*c* and as (2*c*)-1, where *c* is the card number. For example, the card 3 SMT indices are 5 and 6, and the card 4 SMT indices are 7 and 8.
- The MAC for each SMT has 1 as its index. For example, the MAC on the first SMT on card 3 has 5.1 as its index.
- The two token-ring paths on each SMT are indexed 1 and 2. For example, the two paths on the first SMT on card 3 have 5.1 and 5.2 as their indices.
- The two physical ports for the logical interface on each SMT are also indexed 1 and 2. For example, the two ports for the first SMT on card 3 have 5.1 and 5.2 as their indices.

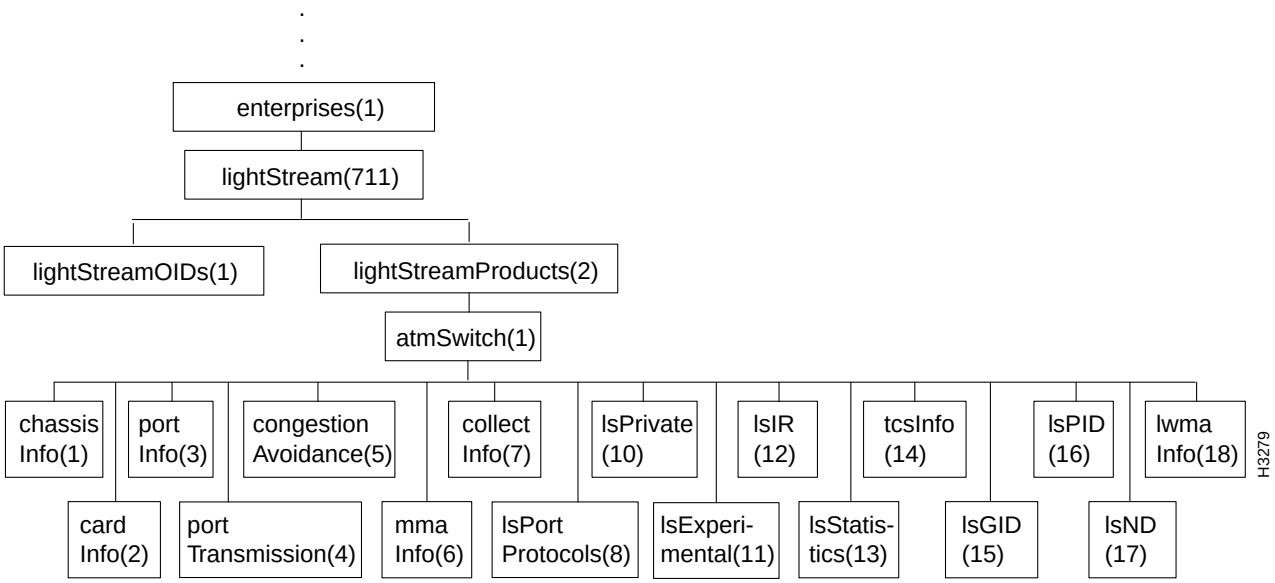
The following example shows the fddimibPATHTable objects for a LightStream node. These objects are defined for card 3 (SMT 5 and 6).

```
cli> walksnmp fddimibPATHTable
Name: fddimibPATHSMTIndex.5.1 Value: 5
Name: fddimibPATHSMTIndex.5.2 Value: 5
Name: fddimibPATHSMTIndex.6.1 Value: 6
Name: fddimibPATHSMTIndex.6.2 Value: 6
Name: fddimibPATHIndex.5.1 Value: 1
Name: fddimibPATHIndex.5.2 Value: 2
Name: fddimibPATHIndex.6.1 Value: 1
Name: fddimibPATHIndex.6.2 Value: 2
Name: fddimibPATHTVXLowerBound.5.1 Value: 2500000
Name: fddimibPATHTVXLowerBound.5.2 Value: 2500000
Name: fddimibPATHTVXLowerBound.6.1 Value: 2500000
Name: fddimibPATHTVXLowerBound.6.2 Value: 2500000
Name: fddimibPATHTMaxLowerBound.5.1 Value: 165000000
Name: fddimibPATHTMaxLowerBound.5.2 Value: 165000000
Name: fddimibPATHTMaxLowerBound.6.1 Value: 165000000
Name: fddimibPATHTMaxLowerBound.6.2 Value: 165000000
Name: fddimibPATHMaxTReq.5.1 Value: 165000000
Name: fddimibPATHMaxTReq.5.2 Value: 165000000
Name: fddimibPATHMaxTReq.6.1 Value: 165000000
Name: fddimibPATHMaxTReq.6.2 Value: 165000000
cli>
```

# The atmSwitch Subtree

Figure 4-3 shows the atmSwitch subtree, which comprises most of the LightStream private MIB.

**Figure 4-3 atmSwitch Subtree Structure**



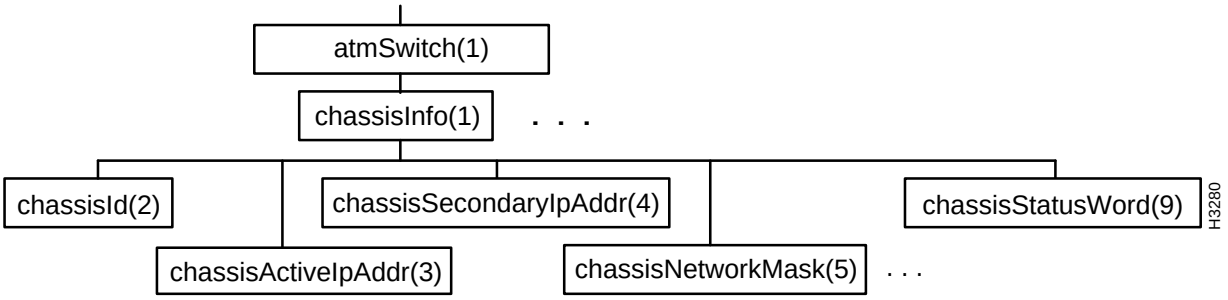

Following are the branches of the atmSwitch subtree:

- The chassisInfo Subtree
- The cardInfo Subtree
- The portInfo Subtree
- The portTransmission Subtree
- The congestionAvoidance Subtree
- The mmaInfo Subtree
- The collectInfo Subtree
- The lsPort Protocols Subtree
- The lsPrivate Subtree
- The lsExperimental Subtree
- The lsIR Subtree
- The lsStatistics Subtree
- The tcsInfo Subtree
- The lsGID Subtree
- The lsPID Subtree
- The lsND Subtree
- The lwmaInfo Subtree

## The chassisInfo Subtree

Figure 4-4 shows the chassisInfo subtree, and Table 4-1 shows the address and value of each object in the subtree.

**Figure 4-4 chassisInfo Subtree Structure**




**Note** Figure 4-4 does not show all the objects in this subtree because there are too many of them. Instead the first and last objects at each level are shown and an ellipsis (...) is used to indicate that additional objects exist. However, all the objects in this subtree are listed in Table 4-1.

**Table 4-1 Objects in the chassisInfo Subtree**

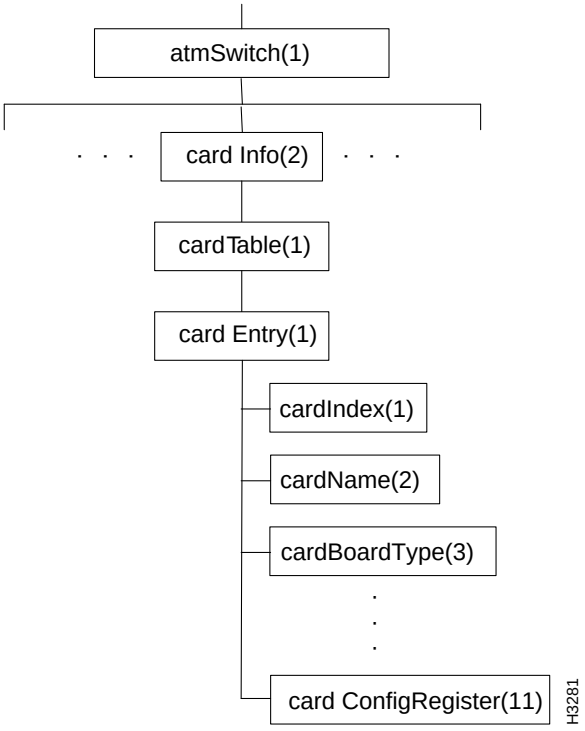
| Objects in the chassisInfo Subtree | Address | Value | |
|---|---|---|---|
| chassisId | chassisInfo.2 | Integer (24 bits) | RO |
| chassisActiveIpAddr | chassisInfo.3 | IpAddress | |
| chassisSecondaryIpAdd | chassisInfo.4 | IpAddress | |
| chassisNetworkMask | chassisInfo.5 | IpAddress | |
| chassisEthernetIpAddr | chassisInfo.6 | IpAddress | |
| chassisEthernetIpMask | chassisInfo.7 | IpAddress | |
| chassisDefaultIpRouter | chassisInfo.8 | IpAddress | |
| chassisStatusWord | chassisInfo.9 | Integer (bitmap) | RO |
| chassisConsoleTrapLevel | chassisInfo.10 | 1 = operational<br>2 = informational<br>3 = trace<br>4 = debug<br>5 = off | RO |

## The cardInfo Subtree

Figure 4-5 shows the cardInfo subtree, and Table 4-2 shows the address and value of each object in the subtree.

The objects in the cardInfo subtree comprise a table, cardTable, in which each cardEntry row contains information about one card in the chassis.

**Figure 4-5 cardInfo Subtree Structure**




**Note** Figure 4-5 does not show all the objects in this subtree because there are too many of them. Instead the first and last objects at each level are shown and an ellipsis (...) is used to indicate that additional objects exist. However, all the objects in this subtree are listed in Table 4-2.

**Table 4-2 cardTable Objects**

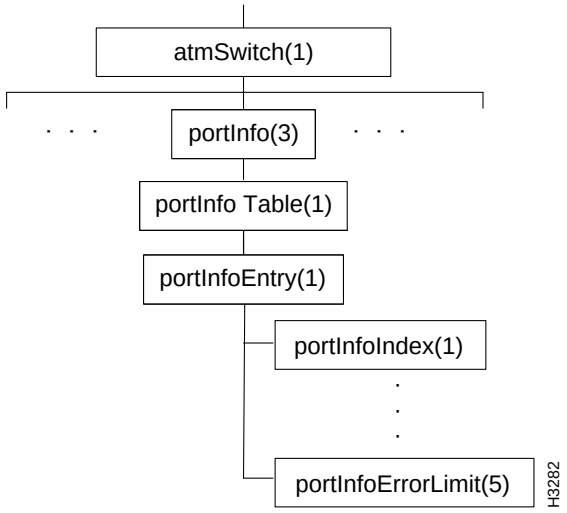
| cardTable Objects | Address | Value | |
|---|---|---|---|
| cardTable | cardInfo.1 | | |
| cardEntry | cardTable.1 | | |
| cardIndex | cardEntry.1 | Integer (card #) | RO |
| cardName | cardEntry.2 | DisplayString | |
| cardBoardType | cardEntry.3 | DisplayString | RO |
| cardLcSoftwareVersion | cardEntry.4 | DisplayString | RO |
| cardLccSoftwareVersion | cardEntry.5 | DisplayString | RO |
| cardPID | cardEntry.6 | Integer (PID) | RO |
| cardMaxVCs | cardEntry.7 | Integer | |
| cardOperStatus | cardEntry.8 | 1 = up<br>2 = down<br>3 = testing<br>4 = empty | RO |

| cardTable Objects | Address | Value | |
|---|---|---|---|
| cardAdminStatus | cardEntry.9 | 1 = up<br>2 = down<br>3 = testing | |
| cardStatusWord | cardEntry.10 | Integer | RO |
| cardConfigRegister | cardEntry.11 | 1 = up<br>2 = down<br>3 = testing<br>4 = empty | RO |

## The portInfo Subtree

Figure 4-6 shows the portInfo subtree, and Table 4-3 shows the address of each object in the subtree and the type of each object.

**Figure 4-6 portInfo Subtree Structure**




The objects in the portInfo subtree comprise a table, portInfo-Table, in which each portInfoEntry row contains information about one port on a card.

**Table 4-3 PortInfoTable Objects**

| PortInfoTable Objects | Address | Value | |
|---|---|---|---|
| portInfoTable | portInfo.1 | | |
| portInfoEntry | portInfoTable.1 | | |
| portInfoIndex | portInfoEntry.1 | Integer<br>((*c**1000)+*p*+*t*) | RO |

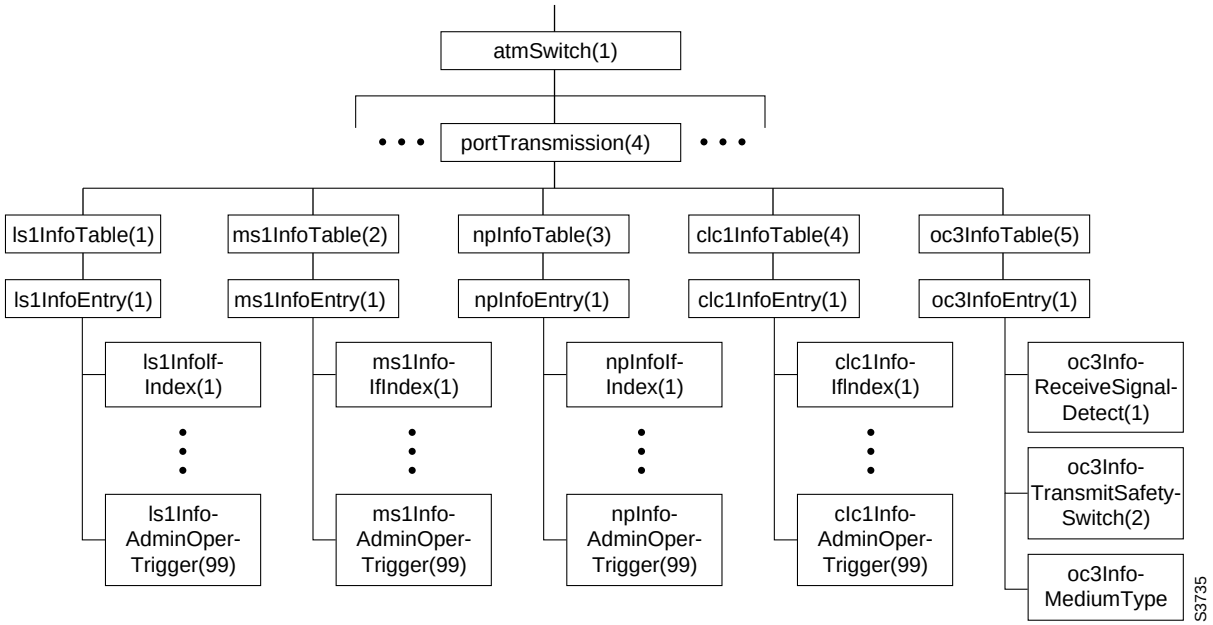
| PortInfoTable Objects | Address | Value | |
|---|---|---|---|
| portInfoType | portInfoEntry.2 | Integer<br>1=empty<br>2=error<br>3=unknown<br>4=switch<br>5=NP<br>6=LS edge<br>7=LS trunk<br>8=MS trunk<br>10=MS edge<br>11=2-pt FDDI<br>12=8-pt Ether<br>13=token ring<br>14=8-pt LS serial edge<br>15=8-pt LS serial trunk<br>30=generic CLC1<br>31=2-pt OC3 edge<br>32=2-pt OC3 trunk<br>33=8-pt T3/E3 edge<br>34=8-pt T3/E3 trunk<br>35=2-pt 155 Mbit TAXI edge<br>36=2-pt 155 Mbit TAXI trunk<br>37=8-pt constant bit rate edge | RO |
| portInfoSpecific | portInfoEntry.3 | ObjectID | RO |
| portInfoName | portInfoEntry.4 | DisplayString | |
| PortInfoErrorLimit | portInfoEntry.5 | Integer ($10^{-n}$ rcv+xmt errors) | |

## The portTransmission Subtree

Figure 4-7 shows the portTransmission subtree, and Table 4-4 through Table 4-8 show the address and type of each object in the subtree.

The objects in the portTransmission subtree comprise three MIB object tables: ls1InfoTable for low speed line cards, ms1InfoTable for medium speed line cards, npInfoTable for NP cards, and clc1InfoTable for cell line cards. Each row of a table (ls1InfoEntry, ms1InfoEntry, or npInfoEntry) contains information about a port on a card.

**Figure 4-7 portTransmission Subtree Structure**




**Note** Figure 4-7 does not show all the objects in this subtree because there are too many of them. Instead the first and last objects at each level are shown and an ellipsis (...) is used to indicate that additional objects exist. However, all the objects in this subtree are listed in Table 4-4 through Table 4-8.

**Table 4-4 ls1InfoTable Objects**

| ls1InfoTable Objects | Address | Value | |
|---|---|---|---|
| ls1InfoTable | portTransmission.1 | | |
| ls1InfoEntry | ls1InfoTable.1 | | |
| ls1InfoIfIndex | ls1InfoEntry.1 | Integer<br>((*c**1000)+*p*+*t*) | RO |
| ls1InfoType | ls1InfoEntry.2 | Integer<br>1=V.35<br>2=RS-422<br>3=RS-530<br>4=T1 | RO |
| ls1InfoOperCsuType | ls1InfoEntry.3 | Integer<br>1=none<br>2=larse | RO |
| ls1InfoAdminCsuType | ls1InfoEntry.4 | Integer<br>1=none<br>2=larse | |

| ls1InfoTable Objects | Address | Value | |
|---|---|---|---|
| ls1InfoOperRcvBaudRate | ls1InfoEntry.5 | Integer | RO |
| ls1InfoAdminRcvBaudRate | ls1InfoEntry.6 | Integer | |
| ls1InfoOperXmitBaudRate | ls1InfoEntry.7 | Integer | RO |
| ls1InfoAdminXmitBaudRate | ls1InfoEntry.8 | Integer | |
| ls1InfoOperNetIntType | ls1InfoEntry.9 | Integer<br>1=DCE<br>2=DTE<br>3=DCE w/local tt clock loop | RO |
| ls1InfoAdminNetIntType | ls1InfoEntry.10 | Integer<br>1=DCE<br>2=DTE<br>3=DCE w/local tt clock loop | |
| ls1InfoOperModemState | ls1InfoEntry.13 | Integer (Bitmask)<br>If port is DTE:<br>0=DCD<br>1=DSR<br>If port is DCE:<br>0=RTS<br>1=DTR | RO |
| ls1InfoOperProtocol | ls1InfoEntry.15 | Integer<br>1 = trunk<br>2 = FR<br>3 = FF<br>4 = PPP<br>5 = unkown | RO |
| ls1InfoAdminProtocol | ls1InfoEntry.16 | Integer<br>1 = trunk<br>2 = FR<br>3 = FF<br>4 = PPP<br>5 = unknown | |
| ls1InfoOperControlBandwidthSize | ls1InfoEntry.21 | Integer | RO |
| ls1InfoAdminControlBandwidthSize | ls1InfoEntry.22 | Integer | |
| ls1InfoOperDataBandwidthSize | ls1InfoEntry.23 | Integer | RO |
| ls1InfoAdminDataBandwidthSize | ls1InfoEntry.24 | Integer | |

| ls1InfoTable Objects | Address | Value | |
|---|---|---|---|
| ls1InfoOperLoopMode | ls1InfoEntry.25 | Integer<br>1 = none<br>2 = internal<br>3 = external<br>4= remote | RO |
| ls1InfoAdminLoopMode | ls1InfoEntry.26 | Integer<br>1 = none<br>2 = internal<br>3 = external<br>4 = remote | |
| ls1InfoLcAutoEnable | ls1InfoEntry.27 | Integer<br>1 = enabled<br>2 = disabled | RO |
| ls1InfoLcDebugLevel | ls1InfoEntry.28 | Integer | RO |
| ls1InfoDataCellCapacity | ls1InfoEntry.29 | Integer | RO |
| ls1InfoDataCellAvailable | ls1InfoEntry.30 | Integer | RO |
| ls1InfoMeasuredBaudRate | ls1InfoEntry.31 | Integer | RO |
| ls1InfoLinkUtilization | ls1InfoEntry.32 | Integer | RO |
| ls1InfoAdminOperTrigger | ls1InfoEntry.99 | Integer<br>1=set admin values<br>99=other | |

**Table 4-5 ms1InfoTable Objects**

| ms1InfoTable Objects | Address | Value | |
|---|---|---|---|
| ms1InfoTable | portTransmission.2 | | |
| ms1InfoEntry | ms1InfoTable.1 | | |
| ms1InfoIfIndex | ms1InfoEntry.1 | Integer<br>((*c**1000)+*p*+*t)* | RO |
| ms1InfoOperCableLength | ms1InfoEntry.2 | Integer<br>1=0-450 ft T3<br>2=450-900 ft T3<br>3=0-400 ft E3<br>4=300-1000 ft E3<br>5=800-1300 ft E3<br>6=1100-1900 ft E3 | RO |

| ms1InfoTable Objects | Address | Value | |
|---|---|---|---|
| ms1InfoAdminCableLength | ms1InfoEntry.3 | Integer 1=0-450 ft T3<br>2=450-900 ft T3<br>3=0-400 ft E3<br>4=300-1000 ft E3<br>5=800-1300 ft E3<br>6=1100-1900 ft E3 | |
| ms1InfoOperProtocol | ms1InfoEntry.4 | Integer<br>1=trunk<br>2=ATM UNI<br>3=unknown | RO |
| ms1InfoAdminProtocol | ms1InfoEntry.5 | Integer<br>1=trunk<br>2=ATM UNI<br>3=unknown | |
| ms1InfoOperControlBandwidthSize | ms1InfoEntry.10 | Integer | RO |
| ms1InfoAdminControlBandwidthSize | ms1InfoEntry.11 | Integer | |
| ms1InfoOperDataBandwidthSize | ms1InfoEntry.12 | Integer | RO |
| ms1InfoAdminDataBandwidthSize | ms1InfoEntry.13 | Integer | |
| msInfoLcAutoEnable | ms1InfoEntry.14 | Integer<br>1=enabled<br>2=disabled | RO |
| msInfoLcDebugLevel | ms1InfoEntry.15 | Integer | RO |
| ms1InfoOperScramble | ms1InfoEntry.16 | Integer<br>1=enabled<br>2=disabled | RO |
| ms1InfoAdminScramble | ms1InfoEntry.17 | Integer<br>1=enabled<br>2=disabled | |
| ms1InfoDataCellCapacity | ms1InfoEntry.18 | Integer | RO |
| ms1InfoDataCellAvailable | ms1InfoEntry.19 | Integer | RO |
| ms1InfoLinkUtilization | ms1InfoEntry.20 | Integer | RO |
| ms1InfoOperFraming | ms1InfoEntry.21 | Integer<br>1=Cell Payload Scrambling<br>2=T3 HEC<br>3=G.804 | RO |
| ms1InfoAdminOperTrigger | ms1InfoEntry.99 | Integer<br>1=set admin values<br>99=other | |

**Table 4-6 npInfoTable Objects**

| npInfoTable Objects | Address | Value | |
|---|---|---|---|
| npInfoTable | portTransmission.3 | | |
| npInfoEntry | npInfotable.1 | | |
| npInfoIfIndex | npInfoEntry.1 | Integer<br>((*c**1000)+*p*+*t*) | RO |
| npInfoIPCommittedRate | npInfoEntry.5 | Integer | |
| npInfoIPCommittedBurst | npInfoEntry.6 | Integer | |
| npInfoIPExcessRate | npInfoEntry.7 | Integer | |
| npInfoIPExcessBurst | npInfoEntry.8 | Integer | |
| npInfoIPNCircuits | npInfoEntry.9 | Integer | |
| npInfoAdminOperTrigger | npInfoEntry.99 | Integer<br>1=set admin values<br>99=other | |

**Table 4-7 clc1InfoTable Objects**

| clc1InfoTable Objects | Address | Value | |
|---|---|---|---|
| clc1InfoTable | portTransmission.4 | | |
| clc1InfoEntry | clc1InfoTable.1 | | |
| clc1InfoIfIndex | clc1InfoEntry.1 | Integer<br>((*c**1000)+*p*+*t*) | RO |
| clc1InfoOperProtocol | clc1InfoEntry.4 | Integer<br>1=trunk<br>2=ATM UNI<br>3=unknown | RO |
| clc1InfoAdminProtocol | clc1InfoEntry.5 | Integer<br>1=trunk<br>2=ATM UNI<br>3=unknown | |
| clc1InfoOperLoopMode | clc1InfoEntry.6 | Integer<br>1=none<br>2=internal<br>3=external | RO |
| clc1InfoAdminLoopMode | clc1InfoEntry.7 | Integer<br>1=none<br>2=internal<br>3=external | |
| clc1InfoOperControlBandwidthSize | clc1InfoEntry.10 | Integer | RO |
| clc1InfoAdminControlBandwidthSize | clc1InfoEntry.11 | Integer | |
| clc1InfoOperDataBandwidthSize | clc1InfoEntry.12 | Integer | RO |
| clc1InfoAdminDataBandwidthSize | clc1InfoEntry.13 | Integer | |

| clc1InfoTable Objects | Address | Value | |
|---|---|---|---|
| clcInfoLcAutoEnable | clc1InfoEntry.14 | Integer<br>1=enabled<br>2=disabled | RO |
| clcInfoLcDebugLevel | clc1InfoEntry.15 | Integer | RO |
| clc1InfoOperScramble | clc1InfoEntry.16 | Integer<br>1=enabled<br>2=disabled | RO |
| clc1InfoAdminScramble | clc1InfoEntry.17 | Enum. Integer<br>1=enabled<br>2=disabled | |
| clc1InfoDataCellCapacity | clc1InfoEntry.18 | Integer | RO |
| clc1InfoDataCellAvailable | clc1InfoEntry.19 | Integer | RO |
| clc1InfoLinkUtilization | clc1InfoEntry.20 | Integer | RO |
| clc1InfoOperClock | clc1InfoEntry.21 | Integer<br>1=internal<br>2=external | RO |
| clc1InfoAdminClock | clc1InfoEntry.22 | Integer<br>1=internal<br>2=external | |
| clc1InfoAdminOperTrigger | clc1InfoEntry.99 | Integer<br>1=set admin values<br>99=other | |

**Table 4-8 oc3InfoTable Objects**

| oc3InfoTable Objects | Address | Value | |
|---|---|---|---|
| oc3InfoTable | portTransmission.5 | | |
| oc3InfoEntry | oc3Infotable.1 | | |
| oc3InfoReceiveSignalDetect | oc3InfoEntry.1 | Integer<br>1=active<br>2=inactive | RO |
| oc3InfoTransmitSafetySwitch | oc3InfoEntry.2 | Integer<br>1=enabled<br>2=disabled | RO |
| oc3InfoMediumType | oc3InfoEntry.3 | Integer<br>1=SONET<br>2=SDH | |

**Note** sonetMediumType in the standard MIB is a read-only object.

## The congestionAvoidance Subtree

Figure 4-8 shows the congestionAvoidance subtree.

**Figure 4-8 congestionAvoidance Subtree Structure**

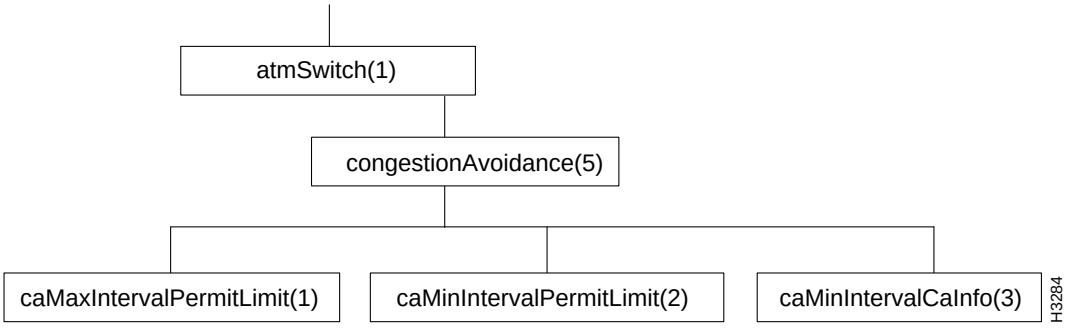



Table 4-9 shows the address and type of each object in the subtree.

**Table 4-9 congestionAvoidance Objects**

| congestionAvoidance Objects | Address | Value |
|---|---|---|
| caMaxIntervalPermitLimit | congestionAvoidance.1 | Integer |
| caMinIntervalPermitLimit | congestionAvoidance.2 | Integer |
| caMinIntervalCaInfo | congestionAvoidance.3 | Integer |

## The mmaInfo Subtree

Figure 4-9 shows the mmaInfo subtree, and the Table 4-10 shows the address and type of each object in the subtree.

The mmacardInfo subtree contains a number of objects and three tables, mmaFtTable, mmaTrapCommunityTable, and mmaNumNameTable.

**Note** Each mmaFtEntry row of mmaFtTable contains information about a current SMTP file transfer. Release 2.0 does not use this function for file transfers; an application would have to be written to use it; this is discouraged.

**Note** Figure 4-9 does not show all the objects in this subtree because there are too many of them. Instead the first and last objects at each level are shown and an ellipsis (...) is used to indicate that additional objects exist. However, all the objects in this subtree are listed in Table 4-10.

**Figure 4-9 mmaInfo Subtree Structure**

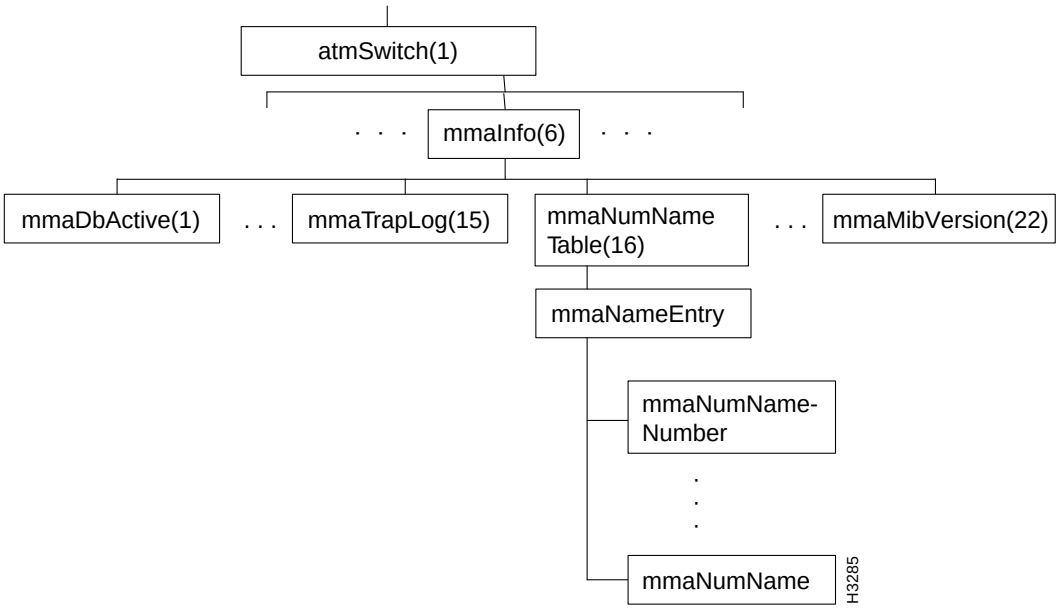



**Table 4-10 mmaInfo Objects**

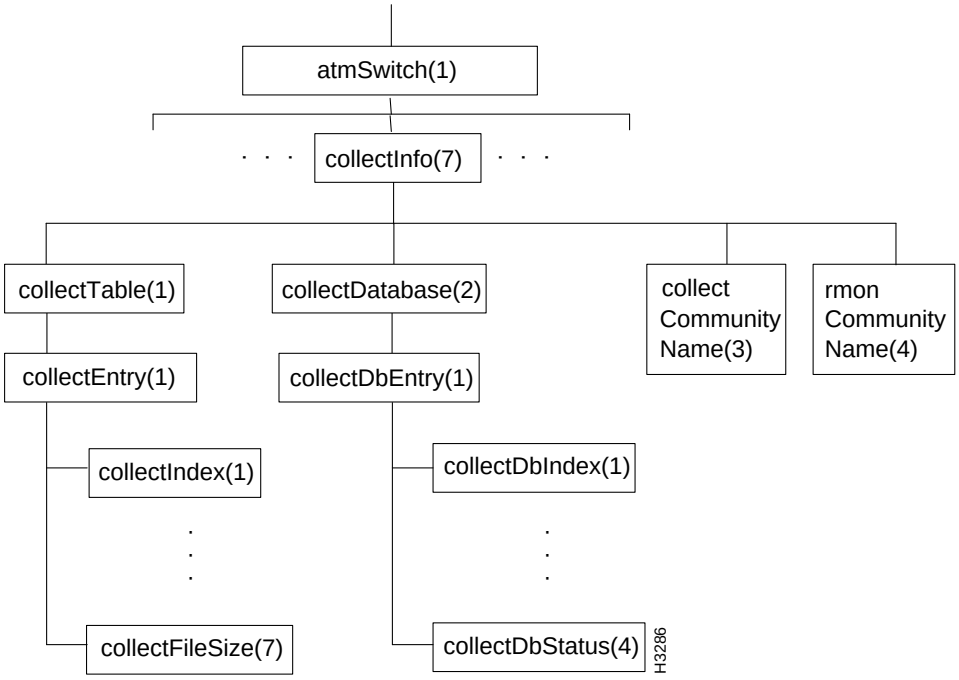
| mmaInfo Objects | Address | Value | |
|---|---|---|---|
| mmaDbActive | mmaInfo.1 | Integer | |
| | | 1=inactive | |
| | | 2=new DB active | |
| mmaTrapFilter | mmaInfo.2 | Integer | |
| | | 1=operational | |
| | | 2=informational | |
| | | 3=trace | |
| | | 4=debug | |
| mmaTrapLanguage | mmaInfo.3 | Integer | |
| | | 1=English | |
| mmaCollectionSpace | mmaInfo.4 | Integer (Kbytes) | |
| mmaConfigHost | mmaInfo.5 | Octet String | |
| mmaConfigAuthor | mmaInfo.6 | Octet String | |
| mmaConfigID | mmaInfo.7 | Integer | |
| mmaSetLock | mmaInfo.8 | Integer | |
| | | 1=unlocked | |
| | | 2=locked (volatile) | |
| | | 3=locked (permanent) | |
| mmaPID | mmaInfo.9 | Integer (PID) | RO |

| mmaInfo Objects | Address | Value | |
|---|---|---|---|
| mmaTrapLog | mmaInfo.10 | Integer<br>1=enabled<br>2=disabled | |
| mmaTrapNumber | mmaInfo.13 | Integer | |
| mmaTrapOnOffState | mmaInfo.14 | Integer<br>1=on<br>2=off<br>3=enable<br>4=disable | |
| mmaNumNameTable | mmaInfo.16 | | |
| mmaNumNameEntry | mmaNumNameTable.1 | | |
| mmaNumNameNumber | mmaNumNameEntry.1 | Integer<br>chassis ID | RO |
| mmaNumName | mmaNumNameEntry.2 | Octet String<br>chassis name | RO |

## The collectInfo Subtree

Figure 4-10 shows the collectInfo subtree, and Table 4-11 shows the address and type of each object in the subtree. The collectInfo subtree contains two objects (collectCommunityName and rmonCommunityName) and two tables, collectTable and collectDatabase. Each row of each table contains information about a collection that has been defined.

**Figure 4-10 collectInfo Subtree Structure**




**Note** Figure 4-10 does not show all the objects in this subtree because there are too many of them. Instead the first and last objects at each level are shown and an ellipsis (...) is used to indicate that additional objects exist. However, all the objects in this subtree are listed in Table 4-11.

**Table 4-11 Objects in the collectInfo Subtree**

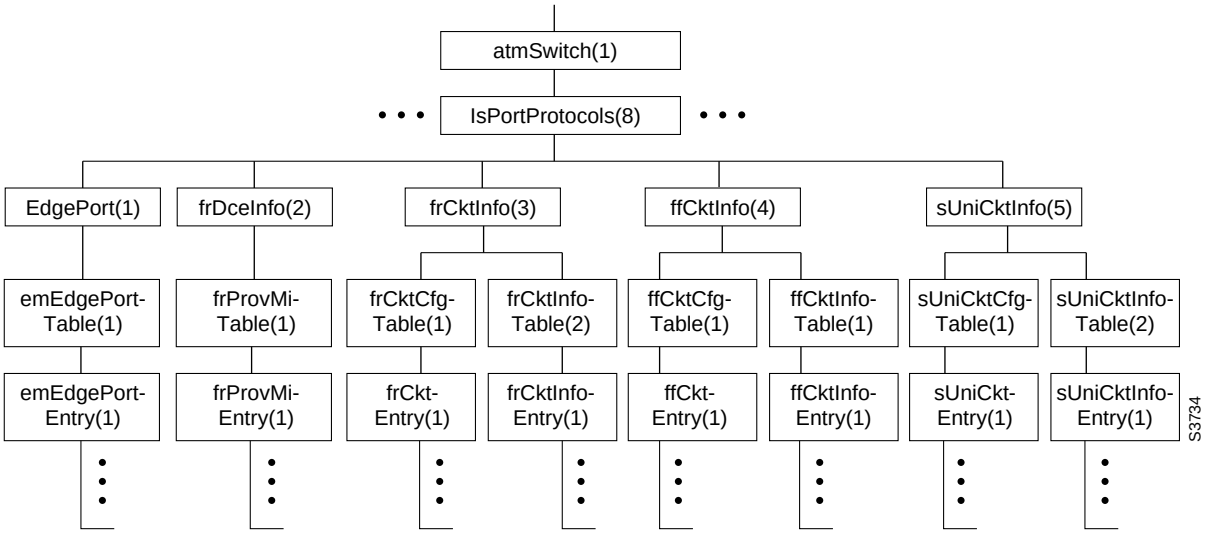
| Objects in the collectInfo Subtree | Address | Value | |
|---|---|---|---|
| collectTable | collectInfo.1 | | |
| collectEntry | collectTable.1 | CollectEntry | |
| collectIndex | collectEntry.1 | Integer<br>collection number | RO |
| collectStatus | collectEntry.2 | Integer<br>1=valid<br>2=create request<br>3=under creation<br>4=invalid | |
| collectStart | collectEntry.3 | Integer (TOD sec) | |
| collectFinish | collectEntry.4 | Integer (TOD sec) | |
| collectInterval | collectEntry.5 | Integer (sec) | |
| collectFileName | collectEntry.6 | DisplayString | RO |
| collectFileSize | collectEntry.7 | Integer (Kbytes) | |

| Objects in the collectInfo Subtree | Address | Value | |
|---|---|---|---|
| collectOperStatus | collectEntry.8 | Integer<br>1=waiting<br>2=running<br>3=under creation | |
| collectDatabase | collectInfo.2 | | |
| collectDbEntry | collectDatabase.1 | | |
| collectDBIndex | collectDbEntry.1 | Integer | RO |
| collectDBInstance | collectDbEntry.2 | Integer | RO |
| collectDBObjectID | collectDbEntry.3 | ObjectID | |
| collectDBStatus | collectDbEntry.4 | Integer<br>1=valid<br>2=create request<br>3=under creation<br>4=invalid | |
| collectCommunityName | collectInfo.3 | DisplayString | |
| rmonCommunityName | collectInfo.4 | DisplayString | |

## The lsPort Protocols Subtree

Figure 4-11 shows the lsPortProtocols subtree, and Table 4-12 through Table 4-19 show the address and type of each object in the subtree. The lsPortProtocols subtree contains several object tables containing information on the various services available on a LightStream network.

**Figure 4-11 lsPort Protocols Subtree Structure**



**Note** Figure 4-11 does not show all the objects in this subtree because there are too many of them. Instead the first and last objects at each level are shown and an ellipsis (...) is used to indicate that additional objects exist. However, all the objects in this subtree are listed in the Table 4-12 through Table 4-19.

**Table 4-12 edgePortTable Objects**

| edgePortTable Objects | Address | Value | |
|---|---|---|---|
| edgePort | lsPortProtocols.1 | | |
| edgePortTable | edgePort.1 | | |
| edgePortEntry | edgePortTable.1 | | |
| edgeIfIndex | edgePortEntry.1 | Integer<br>((*c**1000)+*p*+*t*) | RO |
| edgeUpcType | edgePortEntry.2 | Integer<br>default 1= ANSI | |
| edgeUserDataPerCell | edgePortEntry.3 | Integer (bits) 1-384,<br>default 341 | |
| edgeCellDelayVariance | edgePortEntry.4 | Integer (ms) | |
| edgePrincipalScale | edgePortEntry.5 | Integer<br>default 256 (1:1) | |
| edgeSecondaryScale | edgePortEntry.6 | Integer<br>default 64 (1:4) | |
| edgeMeteringFactor | edgePortEntry.7 | Integer<br>default 64 (*4) | |
| edgeMeteringBurstSize | edgePortEntry.8 | Integer (cells)<br>default 5 | |
| edgeCallSetupRetry | edgePortEntry.9 | Integer (sec)<br>1-3600, default 5 | |
| edgeCallSetupBackoff | edgePortEntry.10 | Integer (exp)<br>1-10, default 5 | |
| edgeMaxFrameSize | edgePortEntry.11 | Integer (bytes)<br>default 1516 | |

**Table 4-13 frProvMiTable Objects**

| frProvMiTable Objects | Address | Value | |
|---|---|---|---|
| frDceInfo | lsPortProtocols.2 | | |
| frProvMiTable | frDceInfo.1 | | |
| frProvMiEntry | frProvMiTable.1 | | |
| frProvMiIfIndex | frProvMiEntry.1 | Index | RO |

| frProvMiTable Objects | Address | Value |
|---|---|---|
| frProvMiState | frProvMiEntry.2 | Integer<br>1=no LMI<br>2=FRIF<br>3=ANSI (default)<br>4=CCITT |
| frProvMiAddressLen | frProvMiEntry.3 | Integer (octets)<br>2=two octets |
| frProvMiNetRequestInterval | frProvMiEntry.4 | Integer (sec)<br>5-30 (default 15) |
| frProvMiNetErrorThreshold | frProvMiEntry.5 | Integer<br>1-10 (default 3) |
| frProvMiNetMonitoredEvents | frProvMiEntry.6 | Integer<br>1-10 (default 4) |
| frProvMiMaxSupportedVCs | frProvMiEntry.7 | Integer |
| frProvMiMulticast | frProvMiEntry.8 | Integer<br>1=non-broadcast |
| frProvMiUserPollingInterval | frProvMiEntry.9 | Integer (sec)<br>5-30 (default 10) |
| frProvMiUserFullEnquiryInterval | frProvMiEntry.10 | Integer<br>1-255 (default 6) |
| frProvMiUserErrorThreshold | frProvMiEntry.11 | Integer<br>1-10 (default 3) |
| frProvMiUserMonitoredEvents | frProvMiEntry.12 | Integer<br>1-10 (default 4) |
| frProvMiNetInterfaceType | frProvMiEntry.13 | Integer<br>1=UNI<br>2=NNI |

**Table 4-14 Objects in the lsPortProtocols Subtree**

| Objects in the lsPortProtocols Subtree | Address | | |
|---|---|---|---|
| frCktInfo | lsPortProtocols.3 | | |
| frCktCfgTable | frCktInfo.1 | | |
| frCktEntry | frCktCfgTable.1 | | |
| frCktSrcNode | frCktEntry.1 | Integer | RO |
| frCktSrcIfIndex | frCktEntry.2 | Integer | RO |
| frCktSrcDlci | frCktEntry.3 | LightStreamDLCI | RO |
| frCktAdminDestNode | frCktEntry.10 | Integer | |
| frCktOperDestNode | frCktEntry.11 | Integer | RO |
| frCktAdminDestIfIndex | frCktEntry.12 | Index | RO |
| frCktOperDestIfIndex | frCktEntry.13 | Index | RO |

| Objects in the lsPortProtocols Subtree | Address | Value | |
|---|---|---|---|
| frCktAdminDestDlci | frCktEntry.14 | LightStreamDLCI | |
| frCktOperDestDlci | frCktEntry.15 | LightStreamDLCI | RO |
| frCktAdminSrcInsuredRate | frCktEntry.25 | Integer (bps)<br>default 0 | |
| frCktOperSrcInsuredRate | frCktEntry.26 | Integer (bps)<br>default 0 | RO |
| frCktAdminSrcInsuredBurst | frCktEntry.27 | Integer (bytes)<br>default 0 | |
| frCktOperSrcInsuredBurst | frCktEntry.28 | Integer (bytes)<br>default 0 | RO |
| frCktAdminSrcMaxRate | frCktEntry.29 | Integer (bps)<br>default: line rate | |
| frCktOperSrcMaxRate | frCktEntry.30 | Integer (bps)<br>default: line rate | RO |
| frCktAdminSrcMaxBurst | frCktEntry.31 | Integer (bytes)<br>default 0 | |
| frCktOperSrcMaxBurst | frCktEntry.32 | Integer (bytes)<br>default 0 | RO |
| frCktOperDestInsuredRate | frCktEntry.34 | Integer (bps)<br>default 0 | RO |
| frCktOperDestInsuredBurst | frCktEntry.36 | Integer (bytes)<br>default 0 | RO |
| frCktOperDestMaxRate | frCktEntry.38 | Integer (bps)<br>default: line rate | RO |
| frCktOperDestMaxBurst | frCktEntry.40 | Integer (bytes)<br>default 0 | RO |
| frCktOperSecondaryScale | frCktEntry.41 | Integer<br>default 255 | RO |
| frCktAdminSecondaryScale | | frCktEntry.42 Integer<br>default 255 | |
| frCktOperPrinBwType | frCktEntry.43 | Integer<br>1=guaranteed (def)<br>2=insured | RO |
| frCktAdminPrinBwType | frCktEntry.44 | Integer<br>1=guaranteed (def)<br>2=insured | |
| frCktOperTransPri | frCktEntry.45 | Integer<br>0, 1 (default 0) | RO |
| frCktAdminTransPri | frCktEntry.46 | Integer<br>0, 1 (default 0) | |

| Objects in the lsPortProtocols Subtree | Address | Value | |
|---|---|---|---|
| frCktOperUserDataPerCell | frCktEntry.47 | Integer (bits)<br>1-384 (default 0) | RO |
| frCktAdminUserDataPerCell | frCktEntry.48 | Integer (bits)<br>1-384 (default 0) | |
| frCktStatus | frCktEntry.99 | Integer<br>1=enabled<br>2=disabled (def)<br>3=delete | |

**Table 4-15 frCktInfoTable Objects**

| | Address | Value | |
|---|---|---|---|
| frCktInfoTable | frCktInfo.2 | | |
| frCktInfoEntry | frCktInfoTable.1 | | |
| frCktInfoIfIndex | frCktInfoEntry.1 | Index | RO |
| frCktInfoDlci | frCktInfoEntry.2 | LightStreamDLCI | RO |
| frCktInfoLclLMI | frCktInfoEntry.3 | Integer<br>1=active<br>2=inactive | RO |
| frCktInfoRmtLMI | frCktInfoEntry.4 | Integer<br>1=active<br>2=inactive | RO |
| frCktInfoCallIDIncoming | frCktInfoEntry.5 | Integer | RO |
| frCktInfoCallIDOutgoing | frCktInfoEntry.6 | Integer | RO |
| frCktInfoDownstreamState | frCktInfoEntry.7 | Integer<br>1=active<br>2=inactive | RO |
| frCktInfoUpstreamState | frCktInfoEntry.8 | Integer<br>1=active<br>2=inactive | RO |
| frCktInfoLastAtmErr | frCktInfoEntry.9 | Octet String | RO |
| frCktInfoDataCellsRequired | frCktInfoEntry.10 | Integer | RO |
| frCktInfoLastAtmLocation | frCktInfoEntry.11 | Octet String | RO |

**Table 4-16 ffCktCfgTable Objects**

| ffCktCfgTable Objects | Address | Value | |
|---|---|---|---|
| ffCktInfo | lsPortProtocols.4 | | |
| ffCktCfgTable | ffCktInfo.1 | | |
| ffCktEntry | ffCktCfgTable.1 | | |
| ffCktSrcNode | ffCktEntry.1 | Integer | RO |

| ffCktCfgTable Objects | Address | Value | |
|---|---|---|---|
| ffCktSrcIfIndex | ffCktEntry.2 | Index (port #) | RO |
| ffCktAdminDestNode | ffCktEntry.9 | Integer | |
| ffCktOperDestNode | ffCktEntry.10 | Integer | RO |
| ffCktAdminDestIfIndex | ffCktEntry.11 | Index (port #) | |
| ffCktOperDestIfIndex | ffCktEntry.12 | Index (port #) | RO |
| ffCktAdminSrcInsuredRate | ffCktEntry.21 | Integer (bps)<br>default 0 | |
| ffCktOperSrcInsuredRate | ffCktEntry.22 | Integer (bps)<br>default -1 | RO |
| ffCktAdminSrcInsuredBurst | ffCktEntry.23 | Integer (bytes)<br>default -1 | |
| ffCktOperSrcInsuredBurst | ffCktEntry.24 | Integer (bytes)<br>default -1 | RO |
| ffCktAdminSrcMaxRate | ffCktEntry.25 | Integer (bps)<br>default: line rate | |
| ffCktOperSrcMaxRate | ffCktEntry.26 | Integer (bps)<br>default: line rate | RO |
| ffCktAdminSrcMaxBurst | ffCktEntry.27 | Integer (bytes)<br>default -1 | |
| ffCktOperSrcMaxBurst | ffCktEntry.28 | Integer (bytes)<br>default -1 | RO |
| ffCktOperDestInsuredRate | ffCktEntry.30 | Integer (bps)<br>default -1 | RO |
| ffCktOperDestInsuredBurst | ffCktEntry.32 | Integer (bytes)<br>default -1 | RO |
| ffCktOperDestMaxRate | ffCktEntry.34 | Integer (bps)<br>default: line rate | RO |
| ffCktOperDestMaxBurst | ffCktEntry.36 | Integer (bytes)<br>default -1 | RO |
| ffCktOperPrinBwType | ffCktEntry.37 | Integer<br>1=guaranteed (def)<br>2=insured | RO |
| ffCktAdminPrinBwType | ffCktEntry.38 | Integer<br>1=guaranteed (def)<br>2=insured | |

| ffCktCfgTable Objects | Address | Value | |
|---|---|---|---|
| ffCktOperTransPri | ffCktEntry.39 | Integer<br>0, 1 (default 1) | RO |
| ffCktAdminTransPri | ffCktEntry.40 | Integer<br>0, 1 (default 1) | |
| ffCktStatus | ffCktEntry.99 | Integer<br>1=enabled<br>2=disabled (def)<br>3=delete | |

**Table 4-17 ffCktInfoTable Objects**

| ffCktInfoTable Objects | Address | Value | |
|---|---|---|---|
| ffCktInfoTable | ffCktInfo.2 | | |
| ffCktInfoEntry | ffCktInfoTable.1 | | |
| ffCktInfoIfIndex | ffCktInfoEntry.1 | Index (port #) | RO |
| ffCktInfoDownstreamState | ffCktInfoEntry.2 | Integer<br>1=active<br>2=inactive | RO |
| ffCktInfoUpstreamState | ffCktInfoentry.3 | Integer<br>1=active<br>2=inactive | RO |
| ffCktInfoCallIDIncoming | ffCktInfoEntry.4 | Integer | RO |
| ffCktInfoCallIDOutgoing | ffCktInfoEntry.5 | Integer | RO |
| ffCktInfoLastAtmErr | ffCktInfoEntry.6 | Octet String | RO |
| ffCktInfoDataCellsRequired | ffCktInfoEntry.7 | Integer (cells) | RO |

**Table 4-18 lsPortProtocols sUniCktInfo Objects**

| lsPortProtocols sUniCktInfo Objects | Address | Value | |
|---|---|---|---|
| sUniCktInfo | lsPortProtocols.5 | | |
| sUniCktCfgTable | sUniCktinfo.1 | | |
| sUniCktEntry | sUniCktCfgTable.1 | | |
| sUniCktSrcNode | sUniCktEntry.1 | Integer | RO |
| sUniCktSrcIfIndex | sUniCktEntry.2 | Index | RO |
| sUniCktSrcVCI | sUniCktEntry.3 | VCI | RO |
| sUniCktAdminDestNode | sUniCktEntry.10 | Integer | |
| sUniCktOperDestNode | sUniCktEntry.11 | Integer | RO |
| sUniCktAdminDestIfIndex | sUniCktEntry.12 | Index | |
| sUniCktOperDestIfIndex | sUniCktEntry.13 | Index | RO |
| sUniCktAdminDestVCI | sUniCktEntry.14 | VCI | |

| IsPortProtocols sUniCktInfo Objects | Address | Value | |
|---|---|---|---|
| sUniCktOperDestVCI | sUniCktEntry.15 | VCI | RO |
| sUniCktOperPrinBwType | sUniCktEntry.22 | Integer<br>1=guaranteed<br>2=insured | RO |
| sUniCktAdminPrinBwType | sUniCktEntry.23 | Integer<br>1=guaranteed<br>2=insured | |
| sUniCktOperTransPri | sUniCktEntry.24 | Integer (0, 1) | RO |
| sUniCktAdminTransPri | sUniCktEntry.25 | Integer (0, 1) | |
| sUniCktAdminSrcInsuredRate | sUniCktEntry.26 | Integer (cps)<br>defaults:<br>MSC: 109<br>CLC: 218 | |
| sUniCktOperSrcInsuredRate | sUniCktEntry.27 | Integer (cps)<br>defaults:<br>MSC: 109<br>CLC: 218 | RO |
| sUniCktAdminSrcInsuredBurst | sUniCktEntry.28 | Integer (cells)<br>default 128 | |
| sUniCktOperSrcInsuredBurst | sUniCktEntry.29 | Integer (cells)<br>default 128 | RO |
| sUniCktAdminSrcMaxRate | sUniCktEntry.30 | Integer (cps)<br>defaults:<br>MSC: 109<br>CLC: 218 | |
| sUniCktOperSrcMaxRate | sUniCktEntry.31 | Integer (cps)<br>defaults:<br>MSC: 109<br>CLC: 218 | RO |
| sUniCktAdminSrcMaxBurst | sUniCktEntry.32 | Integer (cells)<br>default 128 | |
| sUniCktOperSrcMaxBurst | sUniCktEntry.33 | Integer (cells)<br>default 128 | RO |
| sUniCktOperDestInsuredRate | sUniCktEntry.35 | Integer (cps)<br>defaults:<br>MSC: 109<br>CLC: 218 | RO |
| sUniCktOperDestInsuredBurst | sUniCktEntry.37 | Integer (cells)<br>default 128 | RO |

| lsPortProtocols sUniCktInfo Objects | Address | Value | |
|---|---|---|---|
| sUniCktOperDestMaxRate | sUniCktEntry.39 | Integer (cps)<br>defaults:<br>MSC: 109<br>CLC: 218 | RO |
| sUniCktOperDestMaxBurst | sUniCktEntry.41 | Integer (cells)<br>default 128 | RO |
| sUniCktAdminSecondaryScale | sUniCktEntry.42 | Integer<br>default 255 | |
| sUniCktOperSecondaryScale | sUniCktEntry.43 | Integer<br>default 255 | RO |
| sUniCktSts | sUniCktEntry.99 | Integer<br>1=enabled<br>2=disabled (def)<br>3=delete | |

**Table 4-19 sUniCktInfoTable Objects**

| sUniCktInfoTable Objects | Address | Value | |
|---|---|---|---|
| sUniCktInfoTable | sUniCktInfo.2 | | |
| sUniCktInfoEntry | sUniCktInfoTable.1 | | |
| sUniCktInfoIfIndex | sUniCktInfoEntry.1 | Index | RO |
| sUniCktInfoVCI | sUniCktInfoEntry.2 | VCI | RO |
| sUniCktInfoUniToNetCallID | sUniCktInfoEntry.3 | Integer | RO |
| sUniCktInfoNetToUniCallID | sUniCktInfoEntry.4 | Integer | RO |
| sUniCktInfoUniToNetState | sUniCktInfoEntry.5 | Integer<br>1=active | RO |
| sUniCktInfoNetToUniState | sUniCktInfoEntry.6 | Integer<br>2=inactive | RO |
| sUniCktInfoLastAtmErr | sUniCktInfoEntry.7 | Octet String | RO |
| sUniCktInfoDataCellsRequired | sUniCktInfoEntry.8 | Integer (cells) | RO |

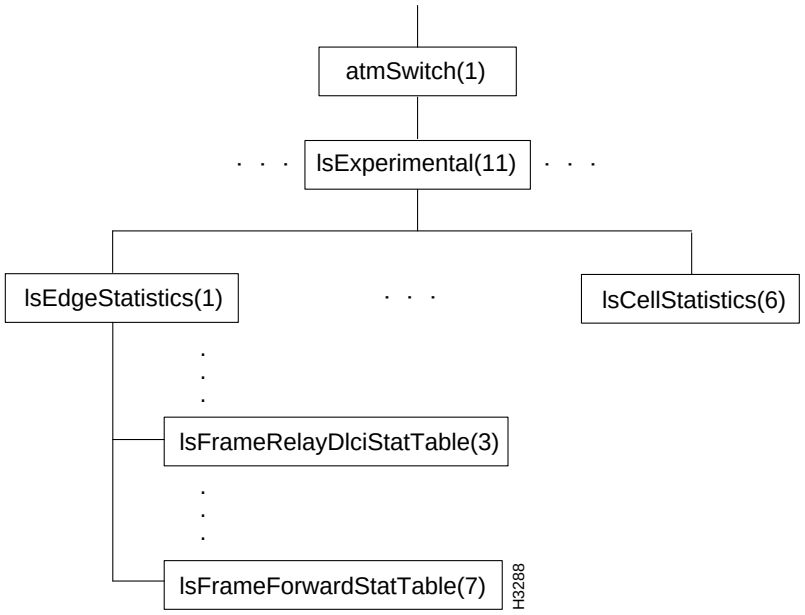
## The lsPrivate Subtree

In Release 2.0, the lsPrivate subtree (atmSwitch.11) is not used and is empty. It is reserved here for use in later releases.

## The lsExperimental Subtree

Figure 4-12 shows portions of the lsExperimental subtree, and Table 4-20 through Table 4-22 show the address and type of each object in the selected portions of the subtree.

**Figure 4-12 lsExperimental Subtree Structure**




In Release 2.0, the lsExperimental subtree contains objects that are used by LightStream to provide advanced support functions and analysis of network performance. This subtree is not used by most users in Release 2.0. In this section, we describe only those MIB objects in the lsExperimental subtree that are used to collect cell statistics and to collect statistics on edge cards for frame relay and frame forwarding. You can use the **getsnmp** command to obtain the statistical information associated with these MIB objects. Ellipses in Figure 4-12 stand for portions of the lsExperimental subtree that are not documented here.

**Table 4-20 lsFrameRelayDlciStatTable Objects**

| lsFrameRelayDlciStatTable Objects | Address | Value | |
|---|---|---|---|
| lsEdgeStatistics | lsExperimentalStatistics.1 | | |
| lsFrameRelayDlciStatTable | lsEdgeStatistics.3 | | |
| lsFrameRelayDlciStatEntry | lsFrameRelayDlciStatTable.1 | | |
| frameRelayDlciStatPortIndex | lsFrameRelayDlciStatEntry.1 | Integer | RO |
| frameRelayDlciStatDlciIndex | lsFrameRelayDlciStatEntry.2 | Integer | RO |
| frameRelayDlciToSwCLP0- Frames | lsFrameRelayDlciStatEntry.3 | Counter | RO |
| frameRelayDlciToSwCLP0Cells | lsFrameRelayDlciStatEntry.4 | Counter | RO |
| frameRelayDlciToSwCLP1- Frames | lsFrameRelayDlciStatEntry.5 | Counter | RO |
| frameRelayDlciToSwCLP1Cells | lsFrameRelayDlciStatEntry.6 | Counter | RO |
| frameRelayDlciToSwDiscard- Frames | lsFrameRelayDlciStatEntry.7 | Counter | RO |
| frameRelayDlciToSwDiscard- Cells | lsFrameRelayDlciStatEntry.8 | Counter | RO |
| frameRelayDlciFrSwCLP0- Frames | lsFrameRelayDlciStatEntry.9 | Counter | RO |

| lsFrameRelayDlciStatTable Objects | Address | Value | |
|---|---|---|---|
| frameRelayDlciFrSwCLP0Cells | lsFrameRelayDlciStatEntry.10 | Counter | RO |
| frameRelayDlciFrSwCLP1- Frames | lsFrameRelayDlciStatEntry.11 | Counter | RO |
| frameRelayDlciFrSwCLP1Cells | lsFrameRelayDlciStatEntry.12 | Counter | RO |

**Table 4-21 lsFrameForwardStatTable Objects**

| lsFrameForwardStatTable Objects | Address | Value | |
|---|---|---|---|
| lsEdgeStatistics | lsExperimentalStatistics.1 | | |
| lsFrameForwardStatTable | lsEdgeStatistics.7 | | |
| lsFrameForwardStatEntry | lsFrameForwardStatTable.1 | | |
| frameForwardStatPortIndex | lsFrameForwardStatEntry.1 | Integer | RO |
| frameForwardToSwCLP0Frames | lsFrameForwardStatEntry.2 | Counter | RO |
| frameForwardToSwCLP0Cells | lsFrameForwardStatEntry.3 | Counter | RO |
| frameForwardToSwCLP1Frames | lsFrameForwardStatEntry.4 | Counter | RO |
| frameForwardToSwCLP1Cells | lsFrameForwardStatEntry.5 | Counter | RO |
| frameForwardToSwDiscard- Frames | lsFrameForwardStatEntry.6 | Counter | RO |
| frameForwardToSwDiscardCells | lsFrameForwardStatEntry.7 | Counter | RO |
| frameForwardFrSwCLP0Frames | lsFrameForwardStatEntry.8 | Counter | RO |
| frameForwardFrSwCLP0Cells | lsFrameForwardStatEntry.9 | Counter | RO |
| frameForwardFrSwCLP1Frames | lsFrameForwardStatEntry.10 | Counter | RO |
| frameForwardFrSwCLP1Cells | lsFrameForwardStatEntry.11 | Counter | RO |

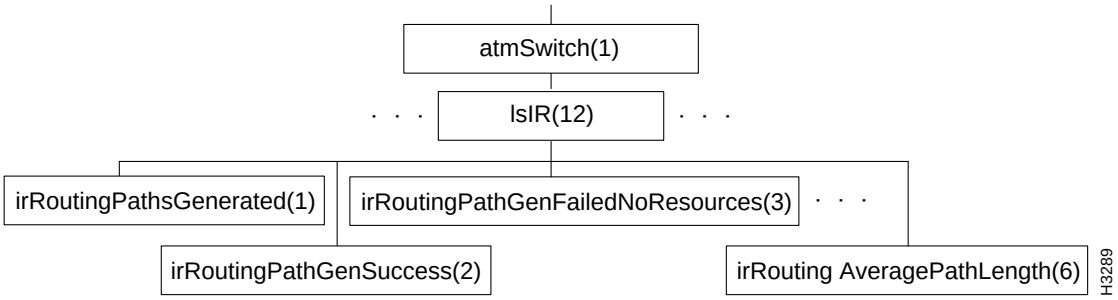
**Table 4-22 lsCellStatistics Objects**

| lsCellStatistics Objects | Address | Value | |
|---|---|---|---|
| lsCellStatistics | lsExperimentalStatistics.6 | | |
| lsCellVciStatTable | lsCellStatistics.1 | | |
| lsCellVciStatEntry | lsCellVciStatTable.1 | | |
| cellVciStatPortIndex | lsCellVciStatEntry.1 | Integer | RO |
| cellVciStatVciIndex | lsCellVciStatEntry.2 | Integer | RO |
| cellVciToSwCLP0Cells | lsCellVciStatEntry.3 | Counter | RO |
| cellVciToSwCLP01Cells | lsCellVciStatEntry.4 | Counter | RO |
| cellVciToSwCLP1Cells | lsCellVciStatEntry.5 | Counter | RO |
| cellVciToSwDiscardCells | lsCellVciStatEntry.6 | Counter | RO |

## The lsIR Subtree

Figure 4-13 shows the internal routing (lsIR) subtree, and Table 4-23 shows the address and type of each object in the lsIR subtree.

**Figure 4-13 lsIR Subtree Structure**




**Note** Figure 4-13 does not show all the objects in this subtree because there are too many of them. Instead the first and last objects at each level are shown and an ellipsis (...) is used to indicate that additional objects exist. However, all the objects in this subtree are listed in Table 4-23.

**Table 4-23 Objects in the lsIR Subtree**

| Objects in the lsIR Subtree | Address | Value | |
|---|---|---|---|
| irRoutingGroup | lsIR.1 | | |
| irRoutingPathsGenerated | irRoutingGroup.1 | Counter | RO |
| irRoutingPathGenSuccess | irRoutingGroup.2 | Counter | RO |
| irRoutingPathGenFailedNoResources | irRoutingGroup.3 | Counter | RO |
| irRoutingPathGenFailedUnknown | irRoutingGroup.4 | Counter | RO |
| irRoutingPathGenFailedOther | irRoutingGroup.5 | Counter | RO |
| irRoutingAveragePathLength | irRoutingGroup.6 | Counter | RO |

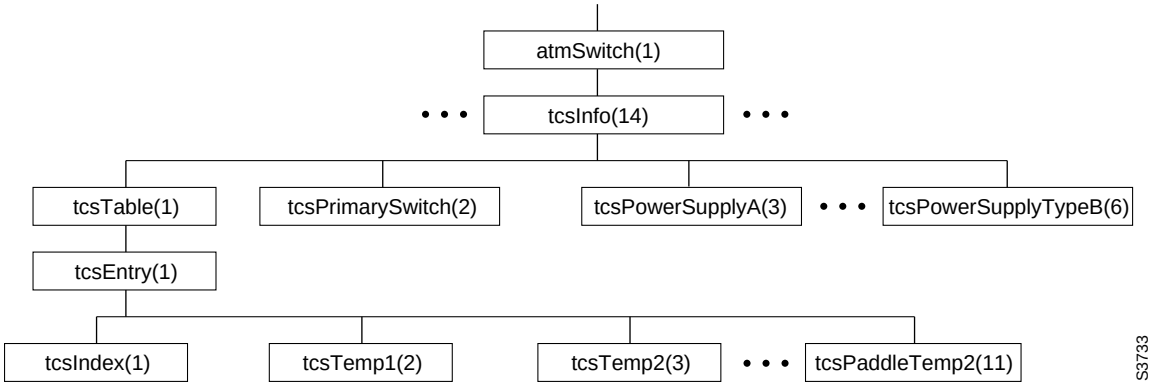
## The lsStatistics Subtree

In Release 2.0, the lsStatistics subtree (atmSwitch.13) is empty. It will be used for objects (presently in the lsExperimental subtree) that LightStream uses to analyze network performance.

## The tcsInfo Subtree

Figure 4-14 shows the tcsInfo subtree, and Table 4-24 shows the address and type of each object in the subtree. The tcsInfo subtree contains one table.

**Figure 4-14 tcsInfo Subtree Structure**




**Note** Figure 4-14 does not show all the objects in this subtree because there are too many of them. Instead the first and last objects at each level are shown and an ellipsis (...) is used to indicate that additional objects exist. However, all the objects in this subtree are listed in Table 4-24.

**Table 4-24 Objects in the tcsInfo Subtree**

| Objects in the tcsInfo Subtree | Address | Value | |
|---|---|---|---|
| tcsTable | tcsInfo.1 | | |
| tcsEntry | tcsTable.1 | | |
| tcsIndex | tcsEntry.1 | Integer | RO |
| tcsTemp1 | tcsEntry.2 | Integer | RO |
| tcsTemp2 | tcsEntry.3 | Integer | RO |
| tcsTcsVoltage | tcsEntry.4 | Integer | RO |
| tcsVccVoltage | tcsEntry.5 | Integer | RO |
| tcsScsiVoltage | tcsEntry.7 | Integer | RO |
| tcsPostResult | tcsEntry.8 | Octet String | RO |

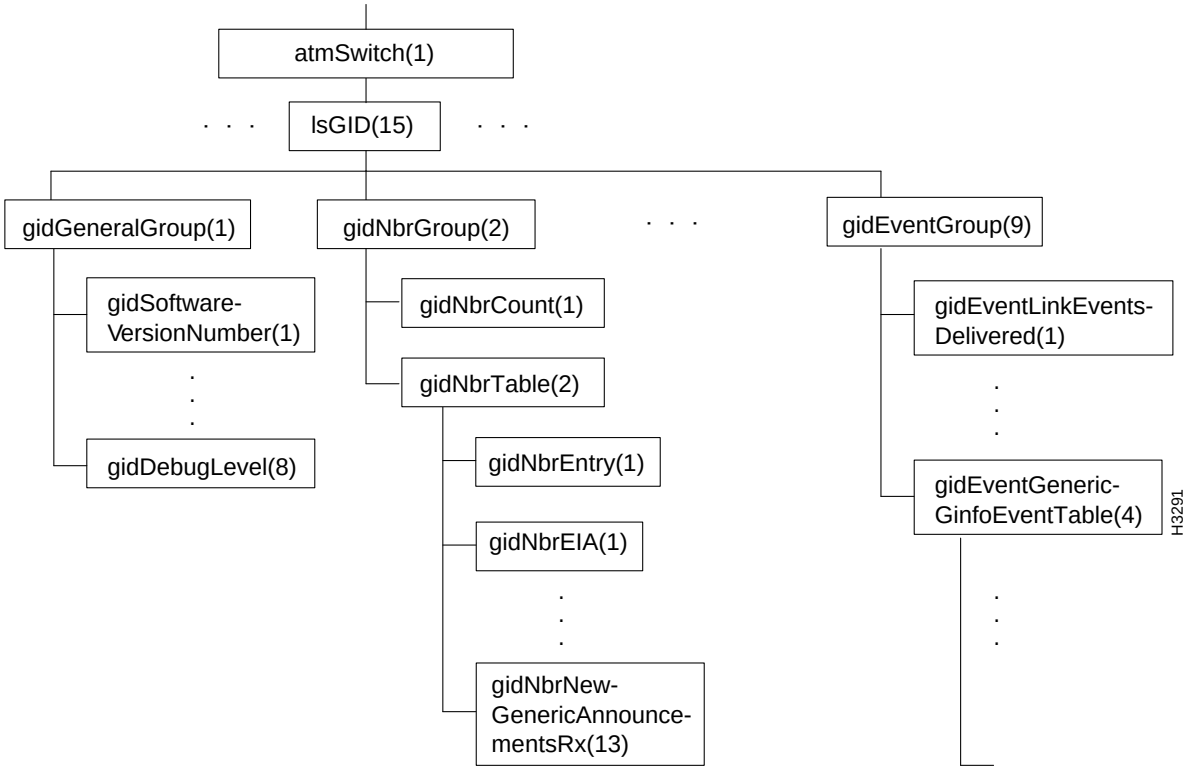
| Objects in the tcsInfo Subtree | Address | Value | |
|---|---|---|---|
| tcsCardType | tcsEntry.9 | Integer<br>1=empty<br>2=error<br>3=unknown<br>4=switch<br>5=NP<br>6=LS edge<br>7=LS trunk<br>8=MS trunk<br>10=MS edge<br>11=2-pt FDDI<br>12=8-pt Ether<br>13=token ring<br>14=8-pt LS ser edge<br>15=8-pt LS ser trunk<br>30=generic CLC1<br>31=2-pt OC3 edge<br>32=2-pt OC3 trunk<br>33=8-pt T3/E3 edge<br>34=8-pt T3/E3 trunk<br>35=2-pt TAXI edge<br>36=2-pt TAXI trunk<br>37=8-pt CBR edge | RO |
| tcsPaddleTemp1 | tcsEntry.10 | Integer | RO |
| tcsPaddleTemp2 | tcsEntry.11 | Integer | RO |
| tcsPaddleWarnTemp1 | tcsEntry.12 | Integer | RO |
| tcsPaddleWarnTemp2 | tcsEntry.13 | Integer | RO |
| tcsPaddleShutdownTemp1 | tcsEntry.14 | Integer | RO |
| tcsPaddleShutdownTemp2 | tcsEntry.15 | Integer | RO |
| tcsWarnTemp1 | tcsEntry.16 | Integer | RO |
| tcsWarnTemp2 | tcsEntry.17 | Integer | RO |
| tcsShutdownTemp1 | tcsEntry.18 | Integer | RO |
| tcsShutdownTemp2 | tcsEntry.19 | Integer | RO |
| tcsFaultLight | tcsEntry.20 | Integer<br>1=on, 2=off | RO |
| tcsReadyLight | tcsEntry.21 | Integer<br>1=on, 2=off | RO |
| tcsSwitchConnectivityMask | tcsEntry.22 | Integer<br>1=on, 2=off | RO |
| tcsPrimarySwitch | tcsInfo.2 | Integer<br>1=A, 2=B | |

| Objects in the tcsInfo Subtree | Address | Value | |
|---|---|---|---|
| tcsPowerSupplyA | tcsInfo.3 | Integer<br>1=empty<br>2=failed<br>3=good | RO |
| tcsPowerSupplyB | tcsInfo.4 | Integer<br>1=empty<br>2=failed<br>3=good | RO |
| tcsPowerSupplyTypeA | tcsInfo.5 | Integer<br>1=empty<br>2=DC power tray<br>3=Todd<br>4=unknown | RO |
| tcsPowerSupplyTypeB | tcsInfo.6 | Integer<br>1=empty<br>2=DC power tray<br>3=Todd<br>4=unknown | RO |
| tcsSwitchFaultMaskA | tcsInfo.7 | Integer<br>bitmask:<br>0: don’t use switch A<br>*n*: slot *n* clock unhealthy | RO |
| tcsSwitchFaultMaskB | tcsInfo.8 | Integer<br>bitmask:<br>0: don’t use switch B<br>*n*: slot *n* clock unhealthy | RO |
| tcsSwitchCutoverSupport | tcsInfo.9 | Integer<br>1=SCs in sync:<br>will do lossless<br>2=SCs not in sync:<br>won’t do lossless<br>3=Other:<br>can’t do lossless | RO |
| tcsFCPrimarySwitchA | tcsInfo.9 | Integer<br>bitmask:<br>if bit *n* = 1 Switch A is supposed to be in slot *n*. | RO |
| tcsSwitchCutoverSupport | tcsInfo.9 | Integer<br>bitmask: if bit *n* = 1 Switch B is supposed to be in slot *n*. | RO |

## The lsGID Subtree

Figure 4-15 shows the IsGID subtree, and Table 4-25 through Table 4-33 show the address and type of each object in the subtree.

**Figure 4-15 IsGID Subtree Structure**




**Note** Figure 4-15 does not show all the objects in this subtree because there are too many of them. Instead, an ellipsis (...) is used to indicate where objects have been omitted from the figure. However, all the objects in this subtree are listed in the Table 4-25 through Table 4-33.

**Table 4-25 gidGeneralGroup Objects**

| gidGeneralGroup Objects | Address | Value | |
|---|---|---|---|
| gidGeneralGroup | lsGID.1 | | |
| gidSoftwareVersionNumber | gidGeneralGroup.1 | DisplayString | RO |
| gidProcessID | gidGeneralGroup.2 | Integer | RO |
| gidUpTime | gidGeneralGroup.3 | Integer (sec) | RO |
| gidMemoryUse | gidGeneralGroup.4 | Counter (bytes) | RO |
| gidTimersProcessed | gidGeneralGroup.5 | Counter | RO |
| gidMallocFailures | gidGeneralGroup.6 | Counter | RO |

**Table 4-26 gidNbrGroup Objects**

| gidNbrGroup Objects | Address | Value | |
|---|---|---|---|
| gidNbrGroup | lsGID.2 | | |
| gidNbrCount | gidNbrGroup.1 | Counter | RO |
| gidNbrTable | gidNbrGroup.2 | | |
| gidNbrEntry | gidNbrTable.1 | | |
| gidNbrEIA | gidNbrEntry.1 | Integer | RO |
| gidNbrVCI | gidNbrEntry.2 | Integer | RO |
| gidNbrState | gidNbrEntry.3 | Integer<br>1=unknown<br>2=exists<br>3=exchange started<br>4=exchange<br>5=loading<br>6=full | RO |
| gidNbrSyncEvents | gidNbrEntry.4 | Counter | RO |
| gidNbrDBReqListLength | gidNbrEntry.5 | Counter | RO |
| gidNbrDBSumListLength | gidNbrEntry.6 | Counter | RO |
| gidNbrHellosRx | gidNbrEntry.7 | Counter | RO |
| gidNbrLinkAnnouncementsRx | gidNbrEntry.8 | Counter | RO |
| gidNbrNewLinkAnnouncementsRx | gidNbrEntry.9 | Counter | RO |
| gidNbrIPAnnouncementsRx | gidNbrEntry.10 | Counter | RO |
| gidNbrNewIPAnnouncementsRx | gidNbrEntry.11 | Counter | RO |
| gidNbrGenericAnnouncementsRx | gidNbrEntry.12 | Counter | RO |
| gidNbrNewGenericAnnouncementsRx | gidNbrEntry.13 | Counter | RO |

**Table 4-27 gidClientGroup Objects**

| gidClientGroup Objects | Address | Value | |
|---|---|---|---|
| gidClientGroup | lsGID.3 | | |
| gidClientCount | gidClientGroup.1 | Counter | RO |
| gidClientTable | gidClientGroup.2 | | |
| gidClientEntry | gidClientTable.1 | | |
| gidClientID | gidClientEntry.1 | Integer | RO |
| gidClientEIA | gidClientEntry.2 | Integer | RO |
| gidClientAnnouncementsRx | gidClientEntry.3 | Counter | RO |
| gidClientLinkAnnouncementsRx | gidClientEntry.4 | Counter | RO |
| gidClientIPAnnouncementsRx | gidClientEntry.5 | Counter | RO |
| gidClientGenericAnnouncementsRx | gidClientEntry.6 | Counter | RO |
| gidClientEventsTx | gidClientEntry.7 | Counter | RO |
| gidClientPathsGenerated | gidClientEntry.8 | Counter | RO |

**Table 4-28 gidIOGroup Objects**

| gidIOGroup Objects | Address | Value | |
|---|---|---|---|
| gidIOGroup | lsGID.4 | | |
| gidIONbrMsgBuffersFree | gidIOGroup.1 | Counter | RO |
| gidIONbrMsgBuffersActive | gidIOGroup.2 | Counter | RO |
| gidIOClientMsgBuffersFree | idIOGroup.3 | Counter | RO |
| gidIOClientMsgBuffersActive | gidIOGroup.4 | Counter | RO |

**Table 4-29 gidFloodGroup Objects**

| gidFloodGroup Objects | Address | Value | |
|---|---|---|---|
| gidFloodGroup | lsGID.5 | | |
| gidFloodClientAnnouncementsRx | gidFloodGroup.1 | Counter | RO |
| gidFloodNbrAnnouncementsRx | gidFloodGroup.2 | Counter | RO |
| gidFloodNbrNewAnnouncementsRx | gidFloodGroup.3 | Counter | RO |
| gidFloodNbrAnnouncementsTx | gidFloodGroup.4 | Counter | RO |
| gidFloodNbrLinkAnnouncementsRx | gidFloodGroup.5 | Counter | RO |
| gidFloodNbrNewLinkAnnouncementsRx | gidFloodGroup.6 | Counter | RO |
| gidFloodNbrLinkAnnouncementsTx | gidFloodGroup.7 | Counter | RO |
| gidFloodNbrIPAnnouncementsRx | gidFloodGroup.8 | Counter | RO |
| gidFloodNbrNewIPAnnouncementsRx | gidFloodGroup.9 | Counter | RO |
| gidFloodNbrIPAnnouncementsTx | gidFloodGroup.10 | Counter | RO |
| gidFloodNbrGenericAnnouncementsRx | gidFloodGroup.11 | Counter | RO |
| gidFloodNbrNewGenericAnnouncementsRx | gidFloodGroup.12 | Counter | RO |
| gidFloodNbrGenericAnnouncementsTx | gidFloodGroup.13 | Counter | RO |

**Table 4-30 gidSyncGroup Objects**

| gidSyncGroup Objects | Address | Value | |
|---|---|---|---|
| gidSyncGroup | lsGID.6 | | |
| gidSyncNbrsExistent | gidSyncGroup.1 | Counter | RO |
| gidSyncNbrsExStart | gidSyncGroup.2 | Counter | RO |
| gidSyncNbrsExchange | gidSyncGroup.3 | Counter | RO |
| gidSyncNbrsLoading | gidSyncGroup.4 | Counter | RO |
| gidSyncNbrsFull | gidSyncGroup.5 | Counter | RO |

**Table 4-31 gidLinkGroup Objects**

| gidLinkGroup Objects | Address | Value | |
|---|---|---|---|
| gidLinkGroup | lsGID.7 | | |
| gidLinkDatabaseSize | gidLinkGroup.1 | Integer (bytes) | RO |
| gidLineCardTable | gidLinkGroup.2 | | |
| gidLineCardEntry | gidLineCardTable.1 | | |
| gidLineCardChassis | gidLineCardEntry.1 | Integer (chassis ID) | RO |
| gidLineCardSlot | gidLineCardEntry.2 | Integer | RO |
| gidLineCardEntryAge | gidLineCardEntry.3 | LightStreamUpToMaxAge | RO |
| gidLineCardEntrySeqno | gidLineCardEntry.4 | Integer | RO |
| gidLineCardEntryAdvNP | gidLineCardEntry.5 | Integer (NP ID) | RO |
| gidLineCardPorts | gidLineCardEntry.6 | Integer | RO |
| gidPortTable | gidLinkGroup.3 | | |
| gidPortEntry | gidPortTable.1 | | |
| gidPortChassis | gidPortEntry.1 | Integer (chassis ID) | RO |
| gidPortID | gidPortEntry.2 | Integer | RO |
| gidPortService | gidPortEntry.3 | Integer<br>1=trunk, 2=edge | RO |
| gidPortUpDown | gidPortEntry.4 | Integer<br>1=down, 2=up | RO |
| gidPortBW0 | gidPortEntry.5 | Integer | RO |
| gidPortBW1 | gidPortEntry.6 | Integer | RO |
| gidPortBW2 | gidPortEntry.7 | Integer | RO |
| gidPortRemoteChassis | gidPortEntry.8 | Integer (chassis ID) | RO |
| gidPortRemotePort | idPortEntry.9 | Integer | RO |

**Table 4-32 gidIpAddressGroup Objects**

| gidIpAddressGroup Objects | Address | Value | |
|---|---|---|---|
| gidIpAddressGroup | lsGID.8 | | |
| gidIpAddressDatabaseSize | gidIpAddressGroup.1 | Integer (bytes) | RO |
| gidIpAddressTable | gidIpAddressGroup.2 | | |
| gidIpAddressEntry | gidIpAddressTable.1 | | |
| GidInternalIpAddress | gidIpAddressEntry.1 | IpAddress | RO |
| gidIpEntryAge | gidIpAddressEntry.2 | LightStreamUpToMaxAge | |
| gidIpEntrySeqno | gidIpAddressEntry.3 | Integer | RO |
| gidIpEntryAdvNP | gidIpAddressEntry.4 | Integer (NP ID) | RO |
| gidIpEntryNetMask | gidIpAddressEntry.5 | Integer | RO |
| gidIpEntryEIA | gidIpAddressEntry.6 | Octet String | RO |

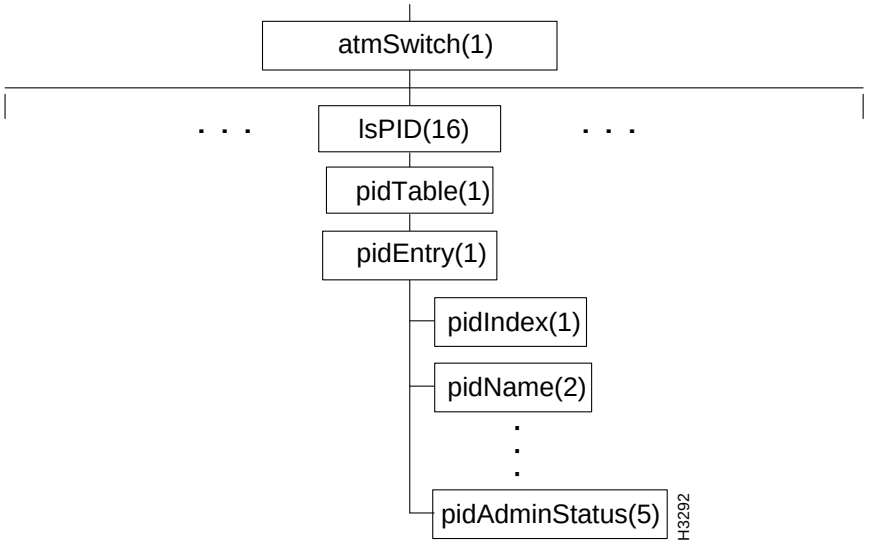
**Table 4-33 gidEventGroup Objects**

| gidEventGroup Objects | Address | Value | |
|---|---|---|---|
| gidEventGroup | lsGID.9 | | |
| gidEventLinkEventsDelivered | gidEventGroup.1 | Counter | RO |
| gidEventIpEventsDelivered | gidEventGroup.2 | Counter | RO |
| gidEventGenericGinfoEventsDelivered | gidEventGroup.3 | Counter | RO |
| gidEventGenericGinfoEventTable | gidEventGroup.4 | | |
| gidEventGenericGinfoEventCount | gidEventGeneric-GinfoEventTable.1 | | |
| gidEventDistributionGroup | gidEventGeneric-GinfoEventCount.1 | Integer | RO |
| gidEventGenericGinfoEvents | gidEventGeneric-GinfoEventCount.2 | Counter | RO |

## The lsPID Subtree

Figure 4-16 shows the lsPID subtree, and Table 4-34 shows the address and type of each object in the subtree.

**Figure 4-16 lsPID Subtree Structure**




**Note** Figure 4-16 does not show all the objects in this subtree because there are too many of them. Instead the first and last objects at each level are shown and an ellipsis (...) is used to indicate that additional objects exist. However, all the objects in this subtree are listed in Table 4-34.

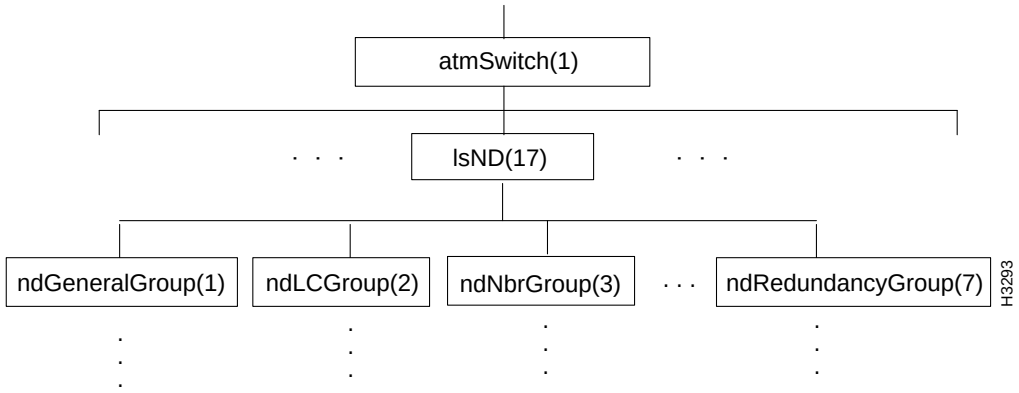
**Table 4-34 Objects in the lsPID Subtree**

| Objects in the lsPID Subtree | Address | Value | |
|---|---|---|---|
| lsPID | atmSwitch.16 | | |
| pidTable | lsPID.1 | | |
| pidEntry | pidTable.1 | | |
| pidIndex | pidEntry.1 | Integer | RO |
| pidName | pidEntry.2 | Integer | RO |
| pidCreationTime | pidEntry.3 | Integer (sec) | RO |
| pidOperStatus | pidEntry.4 | Integer<br>1=up, 2=down | RO |
| pidAdminStatus | pidEntry.5 | Enum. Integer<br>1=up, 2=down | |

## The lsND Subtree

Figure 4-17 shows the lsND subtree, and the Table 4-35 through Table 4-41 show the address and type of each object in the subtree.

**Figure 4-17 lsND Subtree Structure**




**Note** Figure 4-17 does not show all the objects in this subtree because there are too many of them. Instead the first and last objects at each level are shown and an ellipsis (...) is used to indicate that additional objects exist. However, all the objects in this subtree are listed in the Table 4-35 through Table 4-41.

**Table 4-35 ndGeneralGroup Objects**

| ndGeneralGroup Objects | Address | Value | |
|---|---|---|---|
| ndGeneralGroup | lsND.1 | | |
| ndSoftwareVersionNumber | ndGeneralGroup.1 | Display String | RO |

| ndGeneralGroup Objects | Address | Value | |
|---|---|---|---|
| ndProcessID | ndGeneralGroup.2 | Integer | RO |
| ndMemoryUse | ndGeneralGroup.4 | Counter (bytes) | RO |
| ndTimersProcessed | ndGeneralGroup.5 | Counter | RO |

**Table 4-36 ndLCGroup Objects**

| ndLCGroup Objects | Address | Value | |
|---|---|---|---|
| ndLCGroup | lsND.2 | | |
| ndLCCount | ncLCGroup.1 | Counter | RO |
| ndLCTable | ncLCGroup.2 | | |
| ncLCEntry | ndLCTable.1 | | |
| ndLCEIA | ndLCEntry.1 | Integer | RO |
| ndLCChannel | ndLCEntry.2 | Integer | RO |
| ndLCState | ndLCEntry.3 | Integer<br>1=unknown<br>2=exists<br>3=up<br>4=coming down | RO |

**Table 4-37 ndNbrGroup Objects**

| ndNbrGroup Objects | Address | Value | |
|---|---|---|---|
| ndNbrGroup | lsND.3 | | |
| ndNbrCount | ndNbrGroup.1 | Counter | RO |
| ndNbrTable | ndNbrGroup.2 | | |
| ndNbrEntry | ndBbrTable.1 | | |
| ndNbrEIA | ndNbrEntry.1 | Integer | RO |
| ndNbrChannel | ndNbrEntry.2 | Integer | RO |
| ndNbrState | ndNbrEntry.3 | Integer<br>1=unknown<br>2=exists<br>3=up<br>4=coming down | RO |

**Table 4-38 ndSwudGroup Objects**

| ndSwudGroup Objects | Address | Value | |
|---|---|---|---|
| ndSwudGroup | lsND.4 | | |
| ndSwudTable | ndSwudGroup.1 | | |
| ndSwudEntry | ndSwudTable.1 | | |
| ndSwudIndex | ndSwudEntry.1 | Integer | RO |

| ndSwudGroup Objects | Address | Value | |
|---|---|---|---|
| ndOperIntvl | ndSwudEntry.2 | Integer (100ms) | RO |
| ndOperJ | ndSwudEntry.3 | Integer | RO |
| ndOperK | ndSwudEntry.4 | Integer | RO |
| ndOperM | ndSwudEntry.5 | Integer | RO |
| ndOperN | ndSwudEntry.6 | Integer | RO |
| ndAdminIntvl | ndSwudEntry.7 | Integer (100ms) | |
| ndAdminJ | ndSwudEntry.8 | Integer | |
| ndAdminK | ndSwudEntry.9 | Integer | |
| ndAdminM | ndSwudEntry.10 | Integer | |
| ndAdminN | ndSwudEntry.11 | Integer | |
| ndTrigger | ndSwudEntry.12 | Integer | |
| ndSwudStatsInputErrors | ndSwudGroup.2 | Counter | RO |
| ndSwudStatsTable | ndSwudGroup.3 | | |
| NdSwudStatsEntry | ndSwudStatsTable.1 | | |
| ndSwudStatsIndex | ndSwudStatsEntry.1 | Integer | RO |
| ndInputCells | ndSwudStatsEntry.2 | Counter | RO |
| ndInputErrs | ndSwudStatsEntry.3 | Counter | RO |
| ndOutputCells | ndSwudStatsEntry.4 | Counter | RO |
| ndOutputErrs | ndSwudStatsEntry.5 | Counter | RO |

**Table 4-39 ndClient Objects**

| ndClient Objects | Address | Value | |
|---|---|---|---|
| ndClient group | lsND.5 | | |
| ndClientCount | ndClientGroup.1 | Counter | RO |
| ndClientTable | ndClientGroup.2 | | |
| ndClientEntry | ndClientTable.1 | | |
| ndClientID | ndClientEntry.1 | Integer | RO |
| ndClientType | ndClientEntry.2 | Integer<br>3=ND<br>4=GID<br>5=LCC<br>6=CA<br>7=Sys | RO |
| ndClientSubType | ndClientEntry.3 | Integer | RO |
| ndClientEIA | ndClientEntry.4 | Integer | RO |
| ndClientRegistration | ndClientEntry.5 | Integer | RO |

**Table 4-40 ndInternalGroup Objects**

| ndInternalGroup Objects | Address | Value |
|---|---|---|
| ndInternalGroup | lsND.6 | |
| ndInternalDebugLevel | ndInternalGroup.1 | Integer (default 0=off) |
| ndInternalDebugFlags | ndInternalGroup.2 | Integer (default 0=none) |

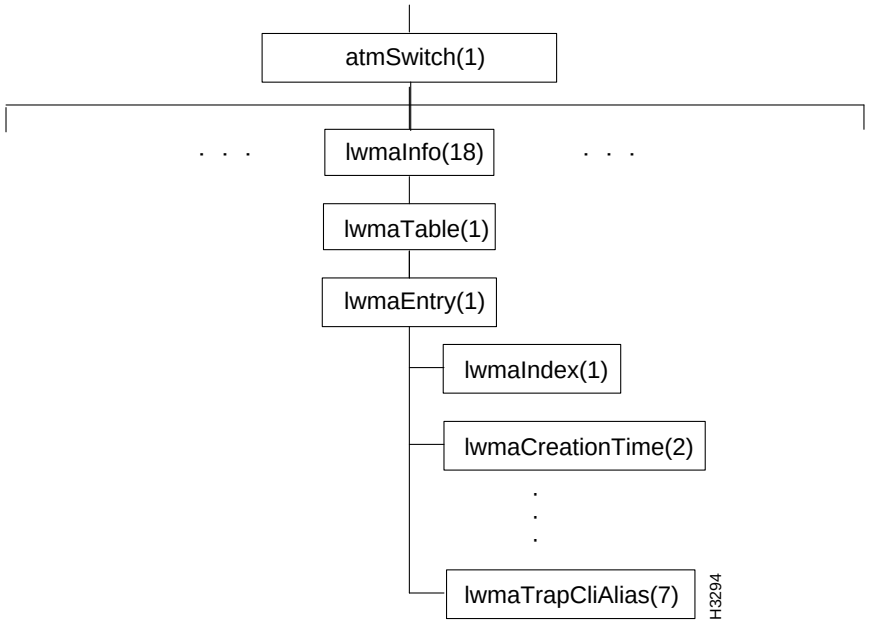
**Table 4-41 ndRedundancyGroup Objects**

| ndRedundancyGroup Objects | Address | Value | |
|---|---|---|---|
| ndRedundancyGroup | lsND.7 | | |
| ndPrimaryNP | ndRedundancyGroup.1 | Integer<br>default 0=none | RO |
| ndThisNP | ndRedundancyGroup.2 | Integer | RO |
| ndForceToBackup | ndRedundancyGroup.3 | Integer | |

## The lwmaInfo Subtree

Figure 4-18 shows the lwmaInfo subtree, and Table 4-42 shows the address and type of each object in the subtree.

**Figure 4-18 lwmaInfo Subtree Structure**




**Note** Figure 4-18 does not show all the objects in this subtree because there are too many of them. Instead the first and last objects at each level are shown and an ellipsis (...) is used to indicate that additional objects exist. However, all the objects in this subtree are listed in Table 4-42.

**Table 4-42 Objects in the lwmaInfo Subtree**

| Objects in the lwmaInfo Subtree | Address | Value | |
|---|---|---|---|
| lwmaTable | lwmaInfo.1 | | |
| lwmaEntry | lwmaTable.1 | | |
| lwmaIndex | lwmaEntry.1 | Integer | RO |
| lwmaCreationTime | lwmaEntry.2 | Integer (sec) | RO |
| lwmaTableNotification | lwmaEntry.3 | Object Identifier | RO |
| lwmaTrapLevel | lwmaEntry.4 | Integer<br>1=operational<br>2=informational<br>3=trace<br>4=debug | |
| lwmaTrapNumber | lwmaEntry.5 | Integer | |
| lwmaTrapOnOffState | lwmaEntry.6 | Integer<br>1=on<br>2=off | |
| lwmaTrapCliAlias | lwmaEntry.7 | Octet String | |

# The lightStreamInternet Subtree

Figure 4-19 shows the lightStreamInternet subtree in the LightStream private MIB. It contains only one branch, the lightStreamBridge branch.

**Figure 4-19 lightStreamInternet Subtree Structure**

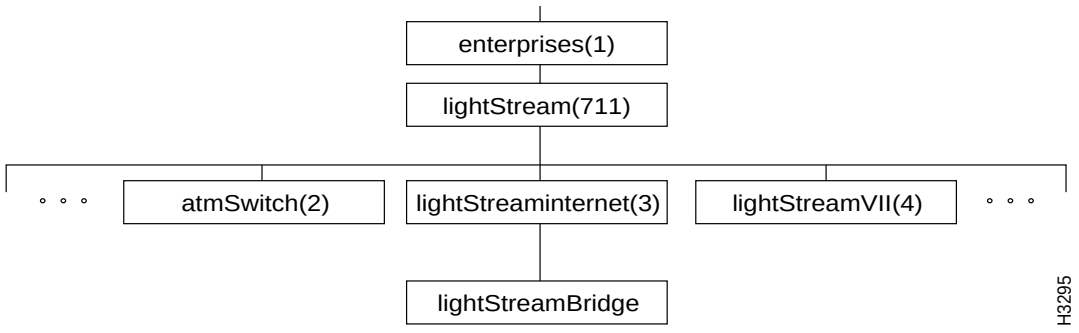
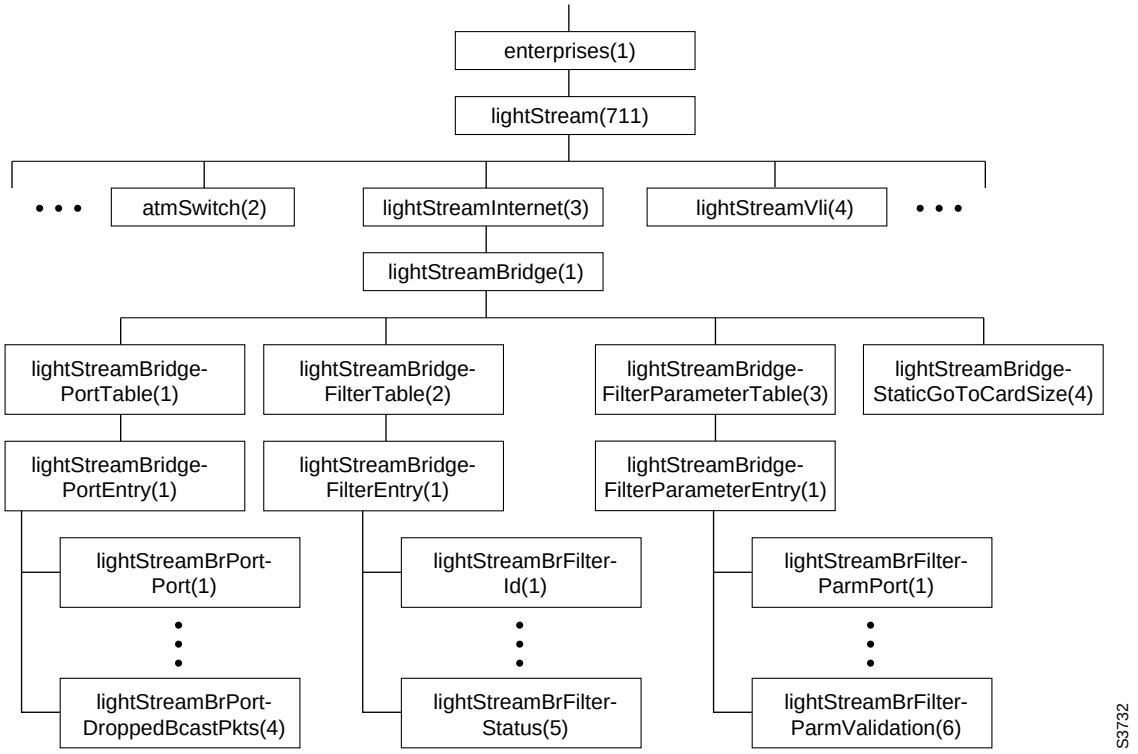



Figure 4-20 shows the lightStreamBridge subtree in the LightStream private MIB, and the Table 4-43 through Table 4-46 show the address and type of each object in the subtree.

**Figure 4-20 lgihtStreamBridge Subtree Structure**




**Note** Figure 4-20 does not show all the objects in this subtree because there are too many of them. Instead the first and last objects at each level are shown and an ellipsis (...) is used to indicate that additional objects exist. However, all the objects in this subtree are listed in the Table 4-43 through Table 4-46.

**Table 4-43 lightStreamBridgePortTable Objects**

| lightStreamBridgePortTable Objects | Address | Value | |
|---|---|---|---|
| lightStreamBridgePortTable | lightStreamBridge.1 | | |
| lightStreamBridgePortEntry | lightStreamBridge PortTable.1 | | |
| lightStreamBrPortPort | lightStreamBridge PortEntry.1 | Integer (dot1dBasePort) | RO |
| lightStreamBrPortDefaultAction | lightStreamBridge PortEntry.2 | 1 = forward<br>2 = block | |
| lightStreamBrPortBcastRateLimit | lightStreamBridge PortEntry.3 | Integer (packets)<br>-1=forward all | |
| lightStreamBrPortDroppedBcastPkts | lightStreamBridge PortEntry.4 | Counter (packets) | RO |

**Table 4-44 lightStreamBridgeFilterTable Objects**

| lightStreamBridgeFilterTable Objects | Address | Value | |
|---|---|---|---|
| lightStreamBridgeFilterTable | lightStreamBridge.1 | | |
| lightStreamBridgeFilterEntry | lightStreamBridge FilterTable.1 | | |
| lightStreamBrFilterId | lightStreamBridge FilterEntry.1 | Integer | RO |
| lightStreamBrFilterTokenIndex | lightStreamBridge FilterEntry.2 | Integer | RO |
| lightStreamBrFilterTokenType | lightStreamBridge | Integer<br>1=operation<br>2=Frame Field<br>3=MAC address<br>4=Ether<br>5=LLC SAP<br>6=LLC Ctl<br>7=SNAP OUI | |
| lightStreamBrFilterTokenValue | lightStreamBridge FilterEntry.4 | DisplayString | |
| lightStreamBrFilterStatus | lightStreamBridge FilterEntry.5 | Integer<br>1=complete<br>2=delete<br>4=intermediate token | |

**Table 4-45 lightStreamBridgeFilterParameterTable Objects**

| lightStreamBridgeFilterParameterTable Objects | Address | Value | |
|---|---|---|---|
| lightStreamBridgeFilterParameterTable | lightStreamBridge.3 | | |
| lightStreamBridgeFilterParameterEntry | lightStreamBridge-FilterParameter Table.1 | | |
| lightStreamBrFilterParmPort | lightStreamBridge FilterParameterEntry.1 | Integer<br>(dot1dBasePort) | RO |
| lightStreamBrFilterParmFilterId | lightStreamBridge FilterParameterEntry.2 | Integer | RO |
| lightStreamBrFilterParmFilterPriority | lightStreamBridge FilterParameterEntry.3 | Integer<br>1=highest<br>default largest + 10 | |
| lightStreamBrFilterParmAction | lightStreamBridge FilterParameterEntry.4 | 1 = forward<br>2 = block | |
| lightStreamBrFilterParmMatchCounts | lightStreamBridge FilterParameterEntry.5 | Counter | RO |
| lightStreamBrFilterParmValidation | lightStreamBridge FilterParameterEntry.6 | Integer<br>1=valid<br>2=invalid | |

**Table 4-46 lightStreamBridgePortTable Objects**

| lightStreamBridgePortTable Objects | Address | Value | |
|---|---|---|---|
| lightStreamBridgeStaticGoToCardSize | lightStreamBridge.4 | Integer<br>(bytes of dot1dStaticGoTo per card) | RO |

# The lightStreamVLI Subtree

Figure 4-21 shows the lightStreamVli subtree in the LightStream private MIB. The objects in this subtree are described in Table 4-47 through Table 4-49.

**Figure 4-21 lightStreamVLI Subtree Structure**

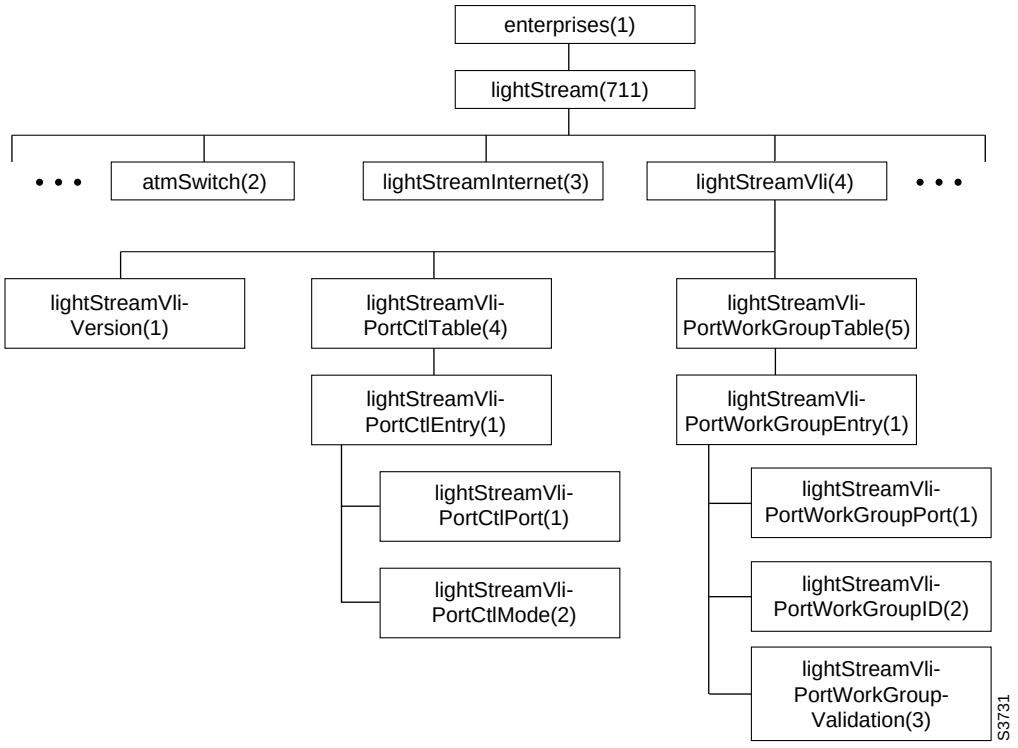



**Table 4-47 VLI Version Object**

| VLI Version Object | Address | Value | |
|---|---|---|---|
| lightStreamVliVersion | lightStreamVli.1 | Integer<br>1=Version 1 | RO |

**Table 4-48 lightStreamVliPortCtlTable Objects**

| lightStreamVliPortCtlTable Objects | Address | Value | |
|---|---|---|---|
| lightStreamVliPortCtlTable | lightStreamVli.4 | | |
| lightStreamVliPortCtlEntry | lightStreamVli PortCtlTable.1 | | |
| lightStreamVliPortCtlPort | lightStreamVli PortCtlEntry.1 | Integer (dot1dBasePort) | RO |
| lightStreamVliPortCtlMode | lightStreamVli PortCtlEntry.2 | Integer 1=include 2=exclude | |

**Table 4-49 lightStreamVliPortWorkGroup Objects**

| lightStreamVliPortWorkGroup Objects | Address | Value | |
|---|---|---|---|
| lightStreamVliPortWorkGroupTable | lightStreamVli.5 | | |
| lightStreamVliPortWorkGroupEntry | lightStreamVliPort WorkGroupTable.1 | | |
| lightStreamVliPortWorkGroupPort | lightStreamVliPort WorkGroupEntry.1 | Integer (dot1dBasePort) | RO |
| lightStreamVliPortWorkGroupID | lightStreamVliPort WorkGroupEntry.2 | Integer (0-65535) | RO |
| lightStreamVliPortWorkGroupValidation | lightStreamVliPort WorkGroupEntry.3 | Integer 1=valid 2=invalid | |

**5 x 5 x**

# LynxOS Command Reference

This chapter contains the manual pages for a number of LynxOS commands that you may need while operating your network of LightStream 2020 enterprise ATM switches. The commands described here are a subset of all the LynxOS commands. They are the LynxOS commands that you are most likely to use.

The **shell** command on CLI accesses the bash shell so that you can execute LynxOS commands. You can also execute LynxOS commands from the bash shell when you log in as superuser.

The following is a list of all the LynxOS commands that are described in this chapter:

- bash - GNU Bourne-Again SHell
- cat - concatenate files
- cbufpr - print contents of a circular buffer
- cp - copy files
- cs - produce file checksum

- date - display or set current date and time
- fsck - file system check and repair
- ftp - file transfer program
- hostname - print or set name of current host system
- ls - show directory contents and file information
- mkdir - create directories
- mkfs - make a file system
- mv - move or rename files
- passwd - change user password
- ps - display status of current processes
- rdist - maintain identical copies of files over multiple hosts
- rm, rmdir - remove (unlink) files or directories
- tail - print last few lines of a file
- tar - combine files into an archive backup
- telnet - user interface to the TELNET protocol
- test_node - test the functionality of a node
- touch - change the modify date of files
- vi - visual text editor

Note also that numerous commands are built in to the shell (see the “BASH Shell Reference” appendix.). Of particular interest are the commands **cd**, **kill**, and **pwd**. The **help** command displays help information about all the shell builtin commands.

# Name

bash - GNU Bourne-Again SHell

# Synopsis

**bash [***options***] [***file***]**

# Copyright



# Description

Bash is an sh-compatible command language interpreter that executes commands read from the standard input or from a file. Bash also incorporates useful features from the *Korn* and *C* shells (ksh and csh).

Bash is ultimately intended to be a faithful implementation of the IEEE Posix Shell and Tools specification (IEEE Working Group 1003.2).

For the full description of bash, refer to the “BASH Shell Reference” appendix.

# Name

cat - concatenate files

# Synopsis

**cat [-benstuv] [***filename* **...]**

# Description

cat concatenates the contents of one or more files onto the standard output. If no *filename* is given, or if the filename "-" is present in the list, cat copies standard input to standard output. If standard out is not redirected to a file or through a pipe, the effect of cat is to print the files on the current tty.

The output of cat is line buffered if standard output is a tty device. Otherwise, output is buffered in large blocks.

# Options

- **-b**

  Omits line numbers from blank lines; used with the -n option.

- **-e**

  Appends a $ character to each line; used with the -v option to indicate where the end-of-line character is.

- **-n**

  Appends the line number to the beginning of each output line.

- **-s**

  Replaces sequences of blank lines by a single blank line.

- **-t**

  Converts tab characters to the string ^I; used with the -v option.

- **-u**

  Forces unbuffered (character-by-character) output.

- **-v**

  Indicates positions of non-printing characters. Non-printable characters whose ASCII value is less than 32 (40 octal) are output as ^ followed by the value plus 64; thus ASCII 26 (control Z) is printed as ^Z. The delete character, ASCII 127 (177 octal), is printed as ^?. Characters whose value is greater than 127 are printed as M- followed by the printable version of the low 7 bits of the value. Note that printable control characters such as newline or tab are not translated.

# Note

Commands like cat a b > a or cat a b > b do not work as expected and in fact destroy information because the shell opens and resets the output files before cat has a chance to read from the input files.

# See Also

Utility Programs — cp - copy files, pr

# Name

cbufpr - print contents of a circular buffer

# Synopsis

**cbufpr[-all] [-h] [-f] [-level {snmp|oper|info|trace|debug}] [-stat] [-tail [-**<*n*>**]] [-v]** <*file*>

# Description

Display contents of a circular buffer file on a remote NP or on a workstation, where the show file command in the CLI is not available. Files in appropriate circular buffer format include the following:

/usr/tmp/apps.log

/usr/tmp/mma/mma.traplog

/usr/tmp/configure/*.log

# Options

The *file* argument is the name of a log file to be printed. It may be omitted with the **-h** or **-v** switch.

- **-all**

  Accept all formats, including plain file.

- **-f**

  Continue reading from tail of file as new entries are added, rather than exiting. Type ^C to kill the process.

- **-h**

  Report this help message. This is the default, with no argument.

- **-level { snmp | oper | info | trace | debug }**

  Only display traps above the indicated trap level. Useful only with the traplog file.

- **-**<*n*>

  An integer specifying lines to print (approximately).

- **-stat**

  Report the current position of the write pointer. Output visible when used with the -tail switch.

- **-tail**

  Read the last 20 lines of the file, or the last *n* lines specified with the -n switch on the same command.

- **-v**

  Report cbufpr version information.

# Name

cp - copy files

# Synopsis

**cp [-irv]** *file1 file2*

**cp [-irv]** *file1 ... directory*

# Description

cp creates copies of files. The destination file [*file2*] is created if it does not exist, or overwritten if it does. If more than two files are given, then the last filename must be a directory, and the files are copied into it with their original base names. cp does not copy a file to itself.

# Options

- **-i**

  cp prompts the user whenever a file is about to be overwritten. Answering y causes cp to continue; anything else causes cp to skip that copy operation.

- **-r**

  cp recursively copies all subtrees of all *file* arguments which are directories and deposits them into the destination directory. cp flags an error if the destination file is not a directory. Hard and symbolic links are not understood by cp; a complete copy is made based upon each directory entry in the source subtree. To truly duplicate subtrees, complete with link information, see tar.

- **-v**

  Print list of files encountered during a recursive cp.

# Examples

**cp file1 file2** — Simply copies from file1 into file2.

**cp file1 file2** — Simply copies from file1 into file2.

**cp file1 file2 file3 dir1** — Copies file1, file2, and file3 into the directory dir1.

**cp -r dir1 file1 dir2** — Copies all files in dir1 recursively into dir2, and also copies file1 into dir2.

# See Also

Utility Programs — cat - concatenate files, pr, mv - move or rename files

# Name

cs - produce file checksum

# Synopsis

**cs [**-*value***] file**

# Description

cs sums every byte of the named *file*, module 256. If no *value* is given, the result is printed to standard output. If a *value* is given, cs compares it to the actual checksum and exits with status 1 if the values are different, or with status 0 if the values are the same.

# Name

date - display or set current date and time

# Synopsis

**date [-u] [-d***n***] [-z***m***] [***newdate***]**

# Description

date can be used either to set the current system date and time or to display it. If no arguments are given, then date only displays the current date and time.

The format for the *newdate* argument describing a new date and time is as follows:

*yymmddhhmm*[.*ss*]

Here, *yy* is the last two digits of the year, *mm* the number of the month (1-12), *dd* the day number in the month, *hh* the hour (0-23), *mm* the minute, and optionally, *.ss* the second. If the date and seconds parts are omitted, the current values are used.

# Options

- **-u**

  Displays the current date in Greenwich Mean Time (GMT). When setting the current time, this flag prevents local transformations of the given time value.

- **-d***n*

  Sets the daylight saving method to n, which must be one of the following values:

  — 0(no daylight savings)

  — 1(USA), 2 (Australia)

  — 3(East Europe)

  — 4(Central Europe)

  — 5(Western Europe).

- **-z***m*

  Sets the time zone to m, given as minutes west of Greenwich (values for North America are thus positive).

# Examples

For example:

- **date -d1 -u 8612080030** — Sets the date to December 8, 1986 12:30 AM GMT, and the daylight saving method to USA.
- **date -z360** — Leaves the current time the same, but changes the time zone to Central Standard Time.

# Name

fsck - file system check and repair

# Synopsis

**fsck** *block-dev*

# Description

fsck checks and repairs the file system on the named block device. It looks for inconsistencies among the following file system components:

- Inode contents — All of the information recorded in each active inode is checked for consistency. The block pointers are all checked for validity. Cross-checks are made to ensure that blocks are owned by at most one inode. The block and byte sizes recorded in the inode are verified and corrected if necessary.
- Directory structure — Directory contents are checked for correctness and consistency. Reference counts for each inode are re-computed and repaired if necessary.
- Free inode list — The free inode list is rebuilt in reverse order of inode number.
- Free block list — The free block list is rebuilt in order of block number.

Any inconsistencies found are reported and repaired or otherwise dealt with.

If orphaned files exist, a /lost+found directory is created and the original /lost+found, if it exists, is renamed /lost+found[A-z], whichever comes first. The orphaned file is put into /lost+found and its file name is set to its inode number.

# Name

ftp - file transfer program

# Synopsis

**ftp [ -v ] [ -d ] [ -i ] [ -n ] [ -g ] [** *host* **]**

# Description

ftp is the user interface to the ARPANET standard File Transfer Protocol. The program allows a user to transfer files to and from a remote network site.

The client host with which ftp is to communicate may be specified on the command line. If this is done, ftp immediately attempts to establish a connection to an FTP server on that host; otherwise, ftp enters its command interpreter and awaits instructions from the user.

# Options

Options may be specified at the command line, or to the command interpreter.

- **-v**

  (Verbose) forces ftp to show all responses from the remote server, as well as report on data transfer statistics.

- **-n**

  Restrains ftp from attempting “auto-login” upon initial connection. If “auto-login” is enabled, ftp checks the .netrc file in the user’s home directory for an entry describing an account on the remote machine. If no entry exists, ftp uses the login name on the local machine as the user identity on the remote machine, and prompts for a password and, optionally, an account with which to log in.

- **-i**

  Turns off interactive prompting during multiple file transfers.

- **-d**

  Enables debugging.

- **-g**

  Disables file name globbing.

# Commands

When ftp is awaiting commands from the user the prompt ftp> is provided the user. The following commands are recognized by ftp:

- **!**

  Invoke a shell on the local machine.

- **append** *local-file* **[** *remote-file* **]**

  Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file. File transfer uses the current settings for type, format, mode, and structure.

- **ascii**

  Set the file transfer type to network ASCII. This is the default type.

- **bell**

  Arrange that a bell be sounded after each file transfer command is completed.

- **binary**

  Set the file transfer type to support binary image transfer.

- **bye**

  Terminate the FTP session with the remote server and exit ftp.

- **cd** *remote-directory*

  Change the working directory on the remote machine to remote-directory.

- **close**

  Terminate the FTP session with the remote server, and return to the command interpreter.

- **connect host [***port***]**

  A synonym for open.

- **delete** *remote-file*

  Delete the file remote-file on the remote machine.

- **debug [***debug-value***]**

  Toggle debugging mode. If an optional debug-value is specified it is used to set the debugging level. When debugging is on, ftp prints each command sent to the remote machine, preceded by the string ->.

- **dir [** *remote-directory* **] [** *local-file* **]**

  Print a listing of the directory contents in the directory *remote-directory* and, optionally, place the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, output comes to the terminal.

- **form** *format*

  Set the file transfer form to *format*. The default format is ”file”.

- **get** *remote-file* **[** *local-file* **]**

  Retrieve the remote-file and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine. The current settings for type, form, mode, and structure are used while transferring the file.

- **hash**

  Toggle hash-sign (#) printing for each data block transferred. The size of a data block is 1024 bytes.

- **glob**

  Toggle file name globbing. With file name globbing enabled, each local file or pathname is processed for shell metacharacters. These characters include “*”, “?”, “[”, “]”, “ ”, “{”, “}”. Remote files specified in multiple-item commands, e.g. **mput**, are globbed by the remote server. With globbing disabled, all files and pathnames are treated literally.

- **help [** *command* **]**

  Print an informative message about the meaning of *command*. If no argument is given, ftp prints a list of the known commands.

- **lcd [** *directory* **]**

  Change the working directory on the local machine. If no directory is specified, the user’s home directory is used.

- **ls [** *remote-directory* **] [** *local-file* **]**

  Print an abbreviated listing of the contents of a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If no local file is specified, the output is sent to the terminal.

- **mdelete** *remote-files*

Delete the specified files on the remote machine. If globbing is enabled, the specification of remote files is first expanded using ls.

- **mdir** *remote-files local-file*

  Obtain a directory listing of multiple files on the remote machine and place the result in local-file.

- **mget** *remote-files*

  Retrieve the specified files from the remote machine and place them in the current local directory. If globbing is enabled, the specification of remote files is first expanded using ls.

- **mkdir** *directory-name*

  Make a directory on the remote machine.

- **mls** *remote-files local-file*

  Obtain an abbreviated listing of multiple files on the remote machine and place the result in local-file.

- **mode [** *mode-name* **]**

  Set the file transfer mode to mode-name. The default mode is “stream” mode.

- **mput** *local-files*

  Transfer multiple local files from the current local directory to the current working directory on the remote machine.

- **open host [** *port* **]**

  Establish a connection to the FTP server host. An optional port number may be supplied, in which case, ftp attempts to contact an FTP server at that port. If “auto-login” is enabled, ftp also attempts to automatically log the user in to the FTP server (see below).

- **prompt**

  Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off (default), any mget or mput transfers all files.

- **put** *local-file* **[** *remote-file* **]**

  Store a local file on the remote machine. If remote-file is left unspecified, the local file name is used in naming the remote file. File transfer uses the current settings for type, format, mode, and structure.

- **pwd**

  Print the name of the current working directory on the remote machine.

- **quit**

  A synonym for bye.

- **quote** *arg1 arg2* **...**

  The arguments specified are sent, verbatim, to the remote FTP server. A single FTP reply code is expected in return.

- **recv** *remote-file* **[** *local-file* **]**

  A synonym for get.

- **remotehelp [** *command-name* **]**

Request help from the remote FTP server. If a command-name is specified it is supplied to the server as well.

- **rename [***from***] [***to***]**

  Rename the file *from* on the remote machine, to the file *to*.

- **rmdir** *directory-name*

  Delete a directory on the remote machine.

- **send** *local-file* **[** *remote-file* **]**

  A synonym for put.

- **sendport**

  Toggle the use of PORT commands. By default, **ftp** attempts to use a PORT command when establishing a connection for each data transfer. If the PORT command fails, **ftp** uses the default data port. When the use of PORT commands is disabled, no attempt is made to use PORT commands for each data transfer. This is useful for certain FTP implementations which do ignore PORT commands but, incorrectly, indicate that they've been accepted.

- **status**

  Show the current status of ftp.

- **struct [** *struct-name* **]**

  Set the file transfer structure to *struct-name*. By default, "stream" structure is used.

- **tenex**

  Set the file transfer type to that needed to talk to TENEX machines.

- **trace**

  Toggle packet tracing.

- **type [** *type-name* **]**

  Set the file transfer type to type-name. If no type is specified, the current type is printed. The default type is network ASCII.

- **user** *user-name* **[** *password* **] [** *account* **]**

  Identify yourself to the remote FTP server. If the password is not specified and the server requires it, ftp prompts the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user is prompted for it. Unless ftp is invoked with "auto-login" disabled, this process is done automatically on initial connection to the FTP server.

- **verbose**

  Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

- **? [** *command* **]**

  A synonym for help.

Command arguments which have embedded spaces may be quoted with double quote (" ") marks.

# File Naming Conventions

Files specified as arguments to ftp commands are processed according to the following rules.

1. If the file name "-" is specified, then stdin (for reading) or stdout (for writing) is used.
2. If the first character of the file name is "--", the remainder of the argument is interpreted as a shell command. ftp then forks a shell, using popen(3) with the argument supplied, and reads (writes) from the stdout (stdin). If the shell command includes spaces, the argument must be quoted; e.g. "-- ls -lt". A particularly useful example of this mechanism is: "dir --more".
3. Failing the above checks, if globbing is enabled, local file names are expanded according to the rules used in the shell; cf. the glob command.

# File Transfer Conventions

The FTP specification specifies many parameters which may affect a file transfer. The type may be one of ascii, image (binary), ebcdic, and local byte size (for PDP- 10's and PDP-20's mostly). The ftp command supports the ascii and image types of file transfer.

ftp supports only the default values for the remaining file transfer parameters: mode, form, and struct.

# Note

Many FTP server implementation do not support the experimental operations such as print working directory (pwd).

Aborting a file transfer does not work right; if one attempts this the local ftp will likely have to be killed by hand.

# Acknowledgements

This section was developed by the University of California, Berkeley.

This LynxOS component is available only as a part of the Lynx TCP/IP package.

# Name

hostname - print or set name of current host system

# Synopsis

**hostname [** *name* **]**

# Description

The hostname command prints the name of the current host. The optional *name* argument is accepted only with superuser privilege, to set the host name.

# Example

This command is especially useful in the CLI when the target host has been reset and the operator is not certain of the actual host, as follows:

```
*cli> shell hostname
chi7
*cli>
```

## See Also

System calls — gethostname()

## Name

ls - show directory contents and file information

## Synopsis

**ls [-1ACFLRacdfgilnqrstu -T types ]** *file* **...**

## Description

ls writes to standard output information requested by options for each file given. If a file is a directory, then the entire contents of that directory are written. Output is sorted alphabetically by default, but can optionally be sorted by access or modification time. If no file is given, the current directory is assumed.

The list of files is sorted alphabetically; non-directory arguments are always processed before directory arguments.

## Options

- **-1**

  One line of output is generated for each file. This option is assumed when standard output is not to a terminal.

- **-A**

  When listing directory contents, all files are included, including those whose names begin with "." but excluding "." and "..".

- **-C**

  ls lists as many files per output line as possible. This option is assumed when standard output is a terminal.

- **-F**

  ls appends markers to file names in the listing to indicate file type, as follows:

  — **/**

  directory

  — **&**

  named pipe (FIFO)

  — **=**

  socket (not implemented in LynxOS)

  — **@**

  symbolic link

— *****

executable regular file

— **|**

ipc special file

- **-L**

  For each argument or directory element which is a symbolic link, information about the file or directory referenced by the link (instead of the link itself) is used for printing, sorting, and/or marking the output.

- **-R**

  Each subdirectory encountered while scanning a directory is itself examined.

- **-T** *types*

  Only those files are shown whose type is one of those specified by *types*. The types list should be a string of one or more of -, b, c, d, f, l, p, r, s, i, I, and +, as described below.

- **-a**

  When listing directory contents, all files are printed, including files which begin with ".".

- **-c**

  The time value examined for printing (-l) and sorting (-u) is the inode change time instead of the modification time.

- **-d**

  Instructs ls to give information about directory arguments themselves instead of listing the contents of the directories.

- **-f**

  Each argument is interpreted as a directory, and contents are listed in the order encountered (no sort is performed). The -f option disables -l, -t, -s, and -r, and enables -a.

- **-g**

  Group ID of each file is printed (only effective when **-l** is also given).

- **-i**

  The inode number of each file is printed.

- **-l**

  Extended information is given for each file, including file mode, number of links, owner, size in bytes, and time (usually modification time, but optionally access or inode change time). If a file is a special file, the major and minor numbers are given in place of file size. If a file is a symbolic link, then the target of the link is given after the file name.

- **-n**

  Restricts listing to files created on the current system date.

- **-q**

  Forces non-printable characters in file names to be printed as "?". This option is assumed when standard output is a terminal.

- **-r**

Reverses sort order.

- **-s**

  Lists size of each file, in 512-byte blocks.

- **-t**

  Sorts by time rather than by name.

- **-u**

  Uses time of last access rather than modification time for printing (-l) and sorting (-u).

## Mode Information

The file mode given with the sf-l option is printed as an ten character string. The first character gives general file type:

- **-**

  Regular file

- **f**

  Regular file

- **r**

  Regular file

- **b**

  Block special file

- **c**

  Character special file

- **d**

  Directory file

- **p**

  Named pipe (FIFO)

- **l**

  Symbolic link

- **s**

  Socket

- **+**

  Contiguous file

- **i**

  Non-persistent ipc special file

- **I**

  Persistent ipc special file

The next nine characters tell whether or not read, write, and execute permissions are on for the owning user, owning group, and others:

- **r**

  read permission

- **w**

  write permission

- **x**

  execute permission

The permissions are printed in the order read-write-execute; a dash in a position means that permission is off. For directory files, execute permission is interpreted as permission to search the directory. If the “sticky bit” in the mode is turned on (see the system call chmod), the last character of the mode string is printed as t rather than the usual x or -.

## See Also

Utility Programs — chmod, chown, ln, wc

System calls — chmod(), chown(), stat(), utimes()

Library functions — directory(), getpwent(), scandir()

## Name

mkdir - create directories

## Synopsis

**mkdir [ -pf ]** *directory* **...**

## Description

mkdir creates directory files with the given pathnames. The standard entries “.” and “..” are automatically created in the new directory. mkdir must have write permission into the parent directory of the new directories.

## Options

- **-f**

  Suppresses error messages.

- **-p**

  mkdir creates each non-existent directory along each given pathname. Thus the command

  **mkdir -p test1/data/numlists/integers**

  creates the subdirectory integers along with test1, data, and numlists, if any of those do not exist.

## See Also

Utility Programs — rmdir

System call - mkdir()

# Name

mkfs - make a file system

# Synopsis

**mkfs [ -Svo ]** *device* **[ [/]***inodes***]**

# Description

Mkfs creates the data structures that comprise a LynxOS file system on *device*. All data previously on the device are lost.

The amount of space on the device is determined automatically. The size of the inode table (in terms of inodes, not bytes) can be given as *inodes*. If *inodes* is preceded by a slash, then it represents an estimate of the average file size (in blocks) on the device; the size of the inode table is set accordingly. As a default, if *inodes* is not given, mkfs creates one inode for every sixteen free blocks.

# Options

- **-S**

  Creates a super block with appropriate values for the device and inode table sizes, end then exits without affecting other parts of the file system. This operation may be useful in desperate situations, such as when **fsck** reports that the super block is “hopelessly damaged”.

- **-v**

  Reports the size of the device and the inode table. The file system is created as usual.

- **-o**

  Creates an old-type file system. An old-type file system uses old-type disk inodes and uses an old-type superblock. Old-type disk inodes are packed eight per block rather than four per block like newer disk inodes. An old-type superblock differs from a newer superblock in that the magic and time fields are not specified. If the -o option is given, **mkfs** sets the magic field in the superblock to zero to distinguish it from a newer superblock. IMPORTANT: This option should be avoided because support for old-type file systems may be phased out in future releases of LynxOS.

# See Also

Utility Programs — fsck, makeboot, mkpart

File formats — boot, superblock

# Name

mv - move or rename files

# Synopsis

**mv [-fi] [-]** *file1 file2*

**mv [-fi] [-]** *file* **...** *directory*

# Description

In the first form, mv changes the name of *file1* to *file2*. If pathname *file2* describes a different directory than *file1*, then the link in the original directory is removed and a link is created in the new directory. In the second form, each *file* is moved from its former directory into *directory*, retaining its base name. In either form, if the move crosses a file system boundary, mv copies the information from the old file to the new file, then removes the old file.

If the destination file exists, and its permissions disallow writing, mv prompts the user and completes the move only on a response of y.

# Options

- **-**

  Allows first filename to begin with "**-**".

- **-f**

  Force writing over existing files. Overrides the -i switch.

- **-i**

  The user is prompted whenever a destination exists, regardless of permissions. The move is completed only on response y.

# Note

mv does not allow a file to be moved onto itself. The side-effects of cross-file system moves are that the user ID of the copying process becomes the owner of the new file, and all linking information tied to other files in the original file system is lost.

# See Also

Utility Programs — cp, ln

System calls — rename()

# Name

passwd - change user password

# Synopsis

**passwd [** *user* **]**

# Description

passwd replaces the current password recorded in the /etc/passwd file for the current user or for the user named *user*. It prompts the user once for the current password, and then twice for a new one to guard against typographic errors. New passwords must be a minimum of 4 characters long if many different characters are used, 6 if the user decides to use a dull, monocase password.

Passwords are kept in the publicly readable /etc/passwd file in an encrypted form.

# Files

- /etc/passwd — User names and passwords
- /etc/ptemp — Temporary copy of /etc/passwd

# See Also

Utility Programs — login

Library functions — crypt(), getpass()

# Name

ps - display status of current processes

# Synopsis

**ps [ afmnostTx ] [ r [** *delay* **] ] [ -p** *pid* **] [ -u** *uid* **] [ -g** *pgrp* **]**

# Description

The ps command displays the current status of processes in the system. The statistics available for each process are as follows:

- pid — Process ID number.
- tid — Thread ID number.
- ppid — Process ID of the parent process.
- pgrp — Process group number.
- pri — System priority.
- text — Size of text (executable code) segment, in terms of 1 kilobyte units. If the text segment is being shared with another process displayed by ps, the text size is followed by an asterisk.
- stack — Size of the process stack segment.
- data — Size of the process data segment.
- sem — Semaphore ID number on which process is waiting, if any.
- time — User and system time used so far by the process. Given as *s.ss* seconds, *mm*:*ss* minutes and seconds, *hh*+*mm*:*ss* hours, minutes and seconds, or *hhhhh*+*mm* hours and minutes, as space allows.
- dev — Control device of the process, as recorded in /dev.

- flags — Value of threads' flags.
- user — User name (based on effective user ID) of the process owner.
- S — Current process state, one of
  - C — Currently executing process (usually ps itself).
  - R — Ready to run, but not running.
  - S — Suspended by a signal (SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU).
  - W — Waiting on a semaphore for some resource.
  - E — New process awaiting completion of its fork operation.
- name — Pathname of the executing process. Currently, the operating system only maintains the first 32 bytes of the pathname.
- dlim — Current data size resource limit, in terms of 1 kilobyte units.
- d% — Percentage of data size limit currently used.
- slim — Current stack size resource limit.
- s% — Percentage of stack size limit currently used.
- smem — Shared memory pages in use, and number of shared memory regions.

By default, ps displays all processes with the same effective user ID as the current real user ID.

# Options

- **a**

  Includes processes owned by others.

- **f**

  Substitute the flags field for the dev field

- **m**

  Show memory usage display instead of normal system use display. This option provides the dlim, d%, slim, s%, and smem fields but leaves out sem, time, dev, user, and S.

- **n**

  Includes the operating system "null" pseudo-process in the output. The accumulated time for the null process is an indication of system idle time.

- **o**

  Includes the supervisor stack size and process state segment size in the stack and data size values.

- **-p** *pid*

  Includes information for process *pid*.

- **-g** *pgrp*

  Includes information for all processes with process group number *pgrp*.

- **r** *delay*

  Tells ps to repeat indefinitely. The display (selected by the other options) is drawn at the top of the CRT. By default, ps waits 10 seconds between updates; *delay* may be given to increase or decrease the update frequency.

- **s**

  Substitute the sem field for the time field.

- **t**

  Includes thread IDs.

- **T**

  Includes stream tasks.

- **-u** *uid*

  Includes information for all processes owned by user *uid*.

- **x**

  Includes processes, owned by you, with no control device.

## Files

/dev/mem — System memory device, used to get information about processes.

## Diagnostics

Since ps cannot sample and display the status off all processes at the speed at which the processes run, its accuracy is limited.

## See Also

Utility Programs — kill, who

System calls — info()

## Name

rdist - maintain identical copies of files over multiple hosts

## Synopsis

**rdist [ -DF***n* **] [ -A** *num* **] [ -a** *num* **] [ -d** *var=value* **] [ -l** <*local_logopts*> **] [-L**<*remote_logopts*>**] [ -f** *distfile* **] [-M** *maxproc* **] [ -m** *host* **] [ -***odistopts* **] [ -t** *timeout* **] [** *name* **... ]**

**rdist -DFn -c name ... [login@]host[:dest]**

**rdist -Server**

**rdist -V**

## Description

rdist is a program to maintain identical copies of files over multiple hosts. It preserves the owner, group, mode, and mtime of files if possible and can update programs that are executing. rdist reads commands from *distfile* to direct the updating of files and/or directories. If *distfile* is '-', the standard input is used. If no -f option is present, the program looks first for *distfile*, then *Distfile* to use as the input. If no names are

specified on the command line, rdist updates all of the files and directories listed in *distfile*. Otherwise, the argument is taken to be the name of a file to be updated or the label of a command to execute. If label and file names conflict, it is assumed to be a label. These may be used together to update specific files using specific commands.

The -c option forces rdist to interpret the remaining arguments as a small distfile. The equivalent distfile is as follows.

```
( name ... ) -> [login@]host
install [dest] ;
```

The -Server option provides partial backward compatible support for older versions of rdist which used this option to put rdist into server mode. If rdist is started with the -Server command line option, it attempts to exec (run) the old version of rdist. This option only works if rdist was compiled with the location of the old rdist (usually either /usr/ucb/oldrdist or /usr/old/rdist) and that program is available at run time.

rdist uses the rcmd(3) interface to access each target host. rdist attempts to run the command

```
rdistd -S
```

on each target host. rdist does not specify the absolute pathname to rdistd on the target host in order to avoid imposing any policy on where rdistd must be installed on target host. Therefore, rdistd must be somewhere in the $PATH of the user running rdist on the remote (target) host.

# Options

- **-A** *num*

  Set the minimum number of free files (inodes) on a filesystem that must exist for rdist to update or install a file.

- **-a** *num*

  Set the minimum amount of free space (in bytes) on a filesystem that must exist for rdist to update or install a file.

- **-D**

  Enable copious debugging messages.

- **-d** *var=value*

  Define *var* to have *value*. This option is used to define or override variable definitions in the distfile. *value* can be the empty string, one name, or a list of names surrounded by parentheses and separated by tabs and/or spaces.

- **-F**

  Do not fork any child rdist processes. All clients are updated sequentially.

- **-f** *distfile*

  Set the name of the distfile to use to be *distfile*. If *distfile* is specified as ‘‘-’’ (dash) then read from standard input (stdin).

- **-l** *logopts*

  Set local logging options. See the section “Message Logging” for details on the syntax for *logopts*.

- **-L** *logopts*

Set remote logging options. *logopts* is the same as for local logging except the values are passed to the remote server (rdistd). See the section “Message Logging” for details on the syntax for *logopts*.

- **-M** *num*

  Set the maximum number of simultaneously running child rdist processes to *num*. The default is 4.

- **-m** *machine*

  Limit which machines are to be updated. Multiple -m arguments can be given to limit updates to a subset of the hosts listed the distfile.

- **-n**

  Print the commands without executing them. This option is useful for debugging *distfile*.

- **-***odistopts*

  Specify the dist options to enable. *distopts* is a comma seperated list of options from the following list:

  — **verify**

  Verify that the files are up to date on all the hosts. Any files that are out of date are displayed but no files are changed, nor is any mail sent.

  — **whole**

  Whole mode. The whole file name is appended to the destination directory name. Normally, only the last component of a name is used when renaming files. This preserves the directory structure of the files being copied instead of flattening the directory structure. For example, rdisting a list of files such as /path/dir1/f1 and /path/dir2/f2 to /tmp/dir would create files /tmp/dir/path/dir1/f1 and /tmp/dir/path/dir2/f2 instead of /tmp/dir/dir1/f1 and /tmp/dir/dir2/f2.

  — **noexec**

  Automatically exclude executable files that are in a.out(5) format from being checked or updated.

  — **younger**

  Younger mode. Files are normally updated if their mtime and size (see stat(2)) disagree. This option causes rdist not to update files that are younger than the master copy. This can be used to prevent newer copies on other hosts from being replaced. A warning message is printed for files which are newer than the master copy.

  — **compare**

  Binary comparison. Perform a binary comparison and update files if they differ, rather than comparing dates and sizes.

  — **follow**

  Follow symbolic links. Copy the file that the link points to rather than the link itself.

  — **ignlnks**

  Ignore unresolved links. rdist normally tries to maintain the link structure of files being transferred and warns the user if all the links cannot be found.

  — **chknfs**

  Do not check or update files on target host that reside on NFS filesystems.

— **chkreadonly**

Enable check on target host to see if a file resides on a read-only filesystem. If a file does, then no checking or updating of the file is attempted.

— **chksym**

If the target on the remote host is a symbolic link, but is not on the master host, the remote target is left a symbolic link. This behavior is generally considered a bug in the original version of rdist, but is present to allow compatibility with older versions.

— **quiet**

Quiet mode. Files that are being modified are normally printed on standard output. This option suppresses this.

— **remove**

Remove extraneous files. If a directory is being updated, any files that exist on the remote host that do not exist in the master directory are removed. This is useful for maintaining truly identical copies of directories.

— **nochkowner**

Do not check user ownership of files that already exist. The file ownership is only set when the file is updated.

— **nochkgroup**

Do not check group ownership of files that already exist. The file ownership is only set when the file is updated.

— **nodescend**

Do not descend into a directory. Normally rdist recursively checks directories. If this option is enabled, then any files listed in the file list in the distfile that are directories are not recursively scanned. Only the existence, ownership, and mode of the directory are checked.

— **savetargets**

Save files that are updated instead of removing them. Any target file that is updated is first renamed from file to file.OLD.

- **-t** *timeout*

  Set the timeout period (in seconds) for waiting for responses from the remote rdist server. The default is 900 seconds.

- **-V**

  Print version information and exit.

# Message Logging

rdist uses a collection of predefined message facilities that each contain a list of message types specifying which types of messages to send to that facility. The local client (rdist) and the remote server (rdistd) each maintain their own copy of what types of messages to log to what facilities.

The -l logopts option to rdist tells rdist what logging options to use locally. The -L logopts option to rdist tells rdist what logging options to pass to the remote rdistd server.

The *logopts* should be of the form

```
facility=types:facility=types...
```

The valid facility names are:

- **stdout**

  Messages to standard output.

- **file**

  Log to a file. To specify the file name, use the format ‘‘file=*filename=types*’’, e.g.

  ```
  file=/tmp/rdist.log=all,debug
  ```

- **syslog**

  Use the syslogd(8) facility.

- **notify**

  Use the internal rdist notify facility. This facility is used in conjunction with the notify keyword in a distfile to specify what messages are mailed to the notify address.

*types* should be a comma separated list of message types. Each message type specified enables that message level. This is unlike the syslog(3) system facility which uses an ascending order scheme. The following are the valid *types*:

- **change**

  Things that change. This includes files that are installed or updated in some way.

- **info**

  General information.

- **notice**

  General info about things that change. This includes things like making directories which are needed in order to install a specific target, but which are not explicitly specified in the distfile.

- **nerror**

  Normal errors that are not fatal.

- **ferror**

  Fatal errors.

- **warning**

  Warnings about errors which are not as serious as nerror *type* messages.

- **debug**

  Debugging information.

- **all**

  All but debug messages.

Here is a sample command line option:

```
-l stdout=all:syslog=change,notice:file=/tmp/rdist.log=all
```

This entry sets local message logging to have all but debug messages sent to standard output, change and notice messages are sent to syslog(3), and all messages are written to the file /tmp/rdist.log.

# Distfiles

The distfile contains a sequence of entries that specify the files to be copied, the destination hosts, and what operations to perform to do the updating. Each entry has one of the following formats.

```
<variable_name> '=' <name_list>
[ label: ] <source_list> '->' <destination_list> <command_list>
[ label: ] <source_list> '::' <time_stamp_file> <command_list>
```

The first format is used for defining variables. The second format is used for distributing files to other hosts. The third format is used for making lists of files that have been changed since some given date. The source list specifies a list of files and/or directories on the local host which are to be used as the master copy for distribution. The destination list is the list of hosts to which these files are to be copied. Each file in the source list is added to a list of changes if the file is out of date on the host which is being updated (second format) or the file is newer than the time stamp file (third format).

Labels are optional. They are used to identify a command for partial updates.

Newlines, tabs, and blanks are only used as separators and are otherwise ignored. Comments begin with '#' and end with a newline.

Variables to be expanded begin with '$' followed by one character or a name enclosed in curly braces (see the examples at the end).

The source and destination lists have the following format:

```
<name>
```

or

```
'(' <zero or more names separated by white-space> ')'
```

These simple lists can be modified by using one level of set addition or subtraction like this:

```
list '+' list
```

or

```
list '-' list
```

If additional modifications are needed (e.g., ''all servers and client client machines except for the OSF/1 machines'') then the list must be explicitly constructed in steps using "temporary" variables.

The shell meta-characters '[', ']', '{', '}', '*', and '?' are recognized and expanded (on the local host only) in the same way as csh(1). They can be escaped with a backslash. The '~' character is also expanded in the same way as csh but is expanded separately on the local and destination hosts. When the -owhole option is used with a file name that begins with '~', everything except the home directory is appended to the destination name. File names which do not begin with '/' or '~' use the destination user's home directory as the root directory for the rest of the file name.

The command list consists of zero or more commands of the following format.

```
'install'<options>opt_dest_name ';'
'notify'<name_list> ';'
'except'<name_list> ';'
'except_pat'<pattern_list>';'
'special'<name_list>string ';'
'cmdspecial'<name_list>string ';'
```

The install command is used to copy out of date files and/or directories. Each source file is copied to each host in the destination list. Directories are recursively copied in the same way. *Opt_dest_name* is an optional parameter to rename files. If no install command appears in the command list or the destination name is not specified, the source file name is used. Directories in the path name are created if they do not

exist on the remote host. The -odistopts option as specified above under Options, has the same semantics as on the command line except they only apply to the files in the source list. The login name used on the destination host is the same as the local host unless the destination name is of the format *login@host*.

The notify command is used to mail the list of files updated (and any errors that may have occurred) to the listed names. If no @ appears in the name, the destination host is appended to the name (e.g., *name1@host, name2@host, ...*).

The except command is used to update all of the files in the source list except for the files listed in *name_list*. This is usually used to copy everything in a directory except certain files.

The except_pat command is like the except command except that *pattern_list* is a list of regular expressions (see ed(1) for details). If one of the patterns matches some string within a file name, that file is ignored. Note that since '\' is a quote character, it must be doubled to become part of the regular expression. Variables are expanded in pattern list but not shell file pattern matching characters. To include a '$', it must be escaped with '\'.

The special command is used to specify sh(1) commands that are to be executed on the remote host after the file in name list is updated or installed. If the name list is omitted then the shell commands are executed for every file updated or installed. The shell variable FILE is set to the current filename before executing the commands in *string*. *String* starts and ends with '"' and can cross multiple lines in the distfile. Multiple commands to the shell should be separated by ';'. Commands are executed in the user's home directory on the host being updated. The special command can be used to rebuild private databases, etc. after a program has been updated.

The cmdspecial command is similar to the special command, except it is executed only when the entire command is completed instead of after each file is updated. The list of files is placed in the environment variable $FILES. Each file name in $FILES is separated by a ':' (semicolon).

If a hostname ends in a + (plus sign), then the plus is stripped off and NFS checks are disabled. This is equivalent to disabling the -ochknfs option just for this one host.

The following is a small example.

```
HOSTS = ( matisse root@arpa)
FILES = ( /bin /lib /usr/bin /usr/games
/usr/include/{*.h,{stand,sys,vax*,pascal,machine}/*.h}
/usr/lib /usr/man/man? /usr/ucb /usr/local/rdist )
EXLIB = ( Mail.rc aliases aliases.dir aliases.pag crontab dshrc
sendmail.cf sendmail.fc sendmail.hf sendmail.st uucp vfont )
${FILES} -> ${HOSTS}
install -oremove,chknfs ;
except /usr/lib/${EXLIB} ;
except /usr/games/lib ;
special /usr/lib/sendmail "/usr/lib/sendmail -bz" ;
srcs:
/usr/src/bin -> arpa
except_pat ( \\.o\$ /SCCS\$ ) ;
IMAGEN = (ips dviimp catdvi)
imagen:
/usr/local/${IMAGEN} -> arpa
install /usr/local/lib ;
notify ralph ;
${FILES} :: stamp.cory
notify root@cory ;
```

# Files

distfile — input command file

/tmp/rdist* — temporary file for update lists

## See Also

sh(1), csh(1), stat(2), rcmd(3)

## Notes

The following options are still recognized for backwards compatibility:

```
-v -N -O -q -b -r -R -s -w -y -h -i -x
```

## Bugs

Source files must reside on the local host where rdist is executed.

Variable expansion only works for name lists; there should be a general macro facility.

rdist aborts on files which have a negative *mtime* (before Jan 1, 1970).

## Name

rm, rmdir - remove (unlink) files or directories

## Synopsis

**rm [***options***]** *file* **...**

**rmdir** *directory* **...**

## Description

The rm command removes directory entries for files. If an entry was the last link to a file, the file is removed as well. Removal of an entry requires write permission in its directory. If a file is not writeable, and rm is using the current terminal for its standard input, the file permissions are printed on the screen and a response is read from the terminal. rm proceeds to remove the file if the response is y.

The rmdir command removes directories. Each directory must be empty except for the special entries "." and "..".

## Options

- **-**

  Allow all arguments following to be treated as filenames. This option is useful for filenames which begin with the "-" character

- **-f**

  Force removal of unwriteable files.

- **-i**

Ask user's permission every time a file or directory is to be removed. If the user responds with a **y**, the file is removed.

- **-r**

  Recursively remove a directory subtree. rm does not remove a directory unless this option is specified.

## See Also

Utility Programs — ln, mv

System calls — rmdir(), unlink()

## Name

tail - print last few lines of a file

## Synopsis

**tail [** *options* **] [** *file* **]**

## Description

The tail command copies to standard output lines from the end of the named *file*, or from standard input if no *file* is given. By default, the last ten lines are printed.

## Options

- **-f**

  tail prints the tail of the file, then waits for the file to grow, printing each new line as it appears. Useful for watching log files.

- **+***n***[blc]**

  Begin output starting *n* blocks (b), lines (l), or characters (c) from the beginning of the file. Default unit is lines.

- **-***n***[blc]**

  Begin output starting *n* blocks, lines, or characters from the end of the file. Default unit is lines.

- **-r**

  Print lines in reverse order. If no distance is specified, tail prints the entire file in reverse.

## Diagnostics

Strange things may happen if tail is used with character special files.

## See Also

Utility Programs — pg, head

# Name

tar - combine files into an archive backup

# Synopsis

**tar {ctxru} [opbXIBfhmvwk] [***blksize***] [***tape***] [-C** *dir***] [-P** *prefix***]** *files* **...**

# Description

The tar command is historically intended to combine files into an archive on a tape. The archive need not actually be a tape. tar operates according to actions given as a command and zero or more modifiers. Each file is copied to or from the archive; directory files are taken to mean the entire subtree beneath the directory.

When creating an archive, tar provides two special mechanisms to allow the archive to reflect a different directory structure than the source of the archived files. A pair of arguments -C *dir* causes the tar process to switch to directory *dir* before interpreting subsequent filenames. A pair -P *prefix* causes the string *prefix* to be appended to the beginning of each filename as it is recorded in the tar output file. Any number of directory-switch and prefixing commands can be given in the file list. Note that once the tar process has changed its working directory, subsequent relative pathnames given in -C requests are interpreted relative to the then current directory.

# Options

## Commands

The tar command string must be the first argument given. It need not (but can) begin with a dash. The basic operation of tar is dictated by one of the following letters embedded in the string:

- **c**

  Create the output archive anew.

- **r**

  Append files to the end of an existing archive.

- **t**

  Pertinent information about files on an existing archive is printed to the standard output. If no file arguments are given to select files in the archive, all files in the archive are shown.

- **u**

  Each file is appended to the archive only if it is not there already, or if it has been changed since last put there. Note that the older versions of files are not deleted.

- **x**

  Each file is extracted from the tape. If a file is actually a directory, the entire subtree is extracted. Shell wildcard operators are recognized within file arguments.

## Modifiers

The following characters may also appear in the command string, and alter the behavior of the basic tar commands.

- **b**

  Instructs tar to recognize the argument following the key as the blocking factor. Tar constructs the archive as groups of 512-byte records. The default blocking factor is 20 records; the maximum blocking factor is 256 records. Each block is written with one write request, and therefore constitutes a physical record on a raw tape device. The blocksize is determined automatically when reading an archive.

  Note: If you use both the b and the f modifiers, be careful to enter their arguments in the correct order. For example:

  **tar cbf 20 archive**

  or:

  **tar cfb archive 20**

- **B**

  This command has no effect and exists for backward compatibility only. Older versions of tar truncate the final block of an archive so that its blocking factor is not always the same as the other blocks. The B command then forced tar to pad the final block with null records up to the blocking factor. The current version of tar always pads the final block, regardless of the B command.

- **f**

  Instructs tar to recognize the argument following the key as the name of the archive. If omitted, tar writes to the file /dev/tar.default. If the file name given is a single dash ('-'), tar writes to standard output or reads from standard input. The tar command thus provides the method of choice for moving directory subtrees, as follows:

  **cd source ; tar cf - | -( cd dest ; tar xf - )-**

- **h**

  Instructs tar to follow symbolic links when creating an archive. Normally, symbolic links are recorded as such in the output.

- **k**

  Instructs tar not to repair access times of files read during archive creation. Normally, tar ensures that access and modification times of files read during archive creation are not affected by the process.

- **l**

  Causes tar to produce error messages during an extraction when attempts to recreate hard links fail.

- **m**

  Prevents restoration of modification times. The time stamp on extracted files is equal to the time at the inception of the tar process.

- **o**

  Prevents tar from explicitly recording directory files in the archive.

- **X**

  Instructs tar to use POSIX-format tar headers, instead of the (simpler and less powerful) UNIX format headers. LynxOS tar can read either format.

- **I**

Ignore checksum. Use if checksum error is reported and you would like to extract the files anyway.

- **p**

  Causes present umask information to be ignored when restoring modes of extracted files.

- **R**

  Suppresses attempts to rename existing files to resolve type conflicts with the archive (see below).

- **v**

  Sets verbose mode, so that tar prints information about each file processed. When checking the contents of an archive (the t command), the verbose option gives more information about the files. Normally tar operates silently.

- **w**

  Causes tar to request verification of each transaction.

# Conflicts During Extraction

When tar is extracting files from an archive into a file system, it can encounter conflicts in file type between existing files and files in the archive. In such cases, tar attempts to rename the existing file and then retries the extraction. Files are renamed by prefixing a dot and appending a suffix of the following form:

```
,tar-03-Sep-1989-10:42:19
```

In other words, the suffix includes the tag ,tar- followed by the date and time of the extraction. All files renamed in a single run of tar have the same date tag. Because the new file names begin with a dot, they do not appear in directory listings. Take care that old renamed files do not clutter the disk.

Conflicts arise when one of the following is true:

- The file in the archive is a directory, and the existing file is not.
- The file in the archive is a hard or symbolic link, and the existing file is simply present.
- The file in the archive is a regular file, and the existing file is not.
- The file in the archive is a regular file, and the existing file is an executing program.

If the R flag is present in the tar command string, renames are not attempted, and messages to that effect are printed should conflicts arise.

# Examples

To collect all files in a subdirectory called dir into an archive called archive:

**tar cf archive dir**

To print the contents of the archive:

**tar tvf archive**

To extract the files from the archive and create a duplicate of *dir* in the subdirectory backup:

**cd backup**

**tar xf ../archive**

To copy the contents of one directory to another:

**cd fromdir**

**tar cf - | ( cd destdir; tar xf - )**

# See Also

Utility Programs — libr

# Name

telnet - user interface to the TELNET protocol

# Synopsis

**telnet [** *host* **[** *port* **] ]**

# Description

The telnet command is used to communicate with another host using the TELNET protocol. If telnet is invoked without arguments, it enters command mode, indicated by its prompt (telnet>). In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an open command (see below) with those arguments.

Once a connection has been opened, telnet enters an input mode. The input mode entered is either character at a time or line by line, depending on what the remote system supports.

In character at a time mode, most text typed is immediately sent to the remote host for processing.

In line-by-line mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The local echo character (initially ^E) may be used to turn the local echo off and on (this would mostly be used to enter passwords without the password being echoed).

In either mode, if the localchars toggle is TRUE (the default in line mode; see below), the user's quit, intr, and flush characters are trapped locally and sent as TELNET protocol sequences to the remote side. There are options (see toggle autoflush and toggle autosynch below) which cause this action to flush subsequent output to the terminal (until the remote host acknowledges the TELNET sequence) and flush previous terminal input (in the case of quit and intr).

While connected to a remote host, telnet command mode may be entered by typing the telnet escape character (initially ^]). When in command mode, the normal terminal editing conventions are available.

# Commands

The following commands are available. Only enough of each command to uniquely identify it need be typed (this is also true for arguments to the mode, set, toggle, and display commands).

- **open** *host* **[***port***]**

  Open a connection to the named host. If no port number is specified, telnet attempts to contact a TELNET server at the default port. The host specification may be either a host name (see hosts (5)) or an Internet address specified in the dot notation (see inet (3N)).

- **close**

  Close a TELNET session and return to command mode.

- **quit**

  Close any open TELNET session and exit telnet. An end of file (in command mode) also closes a session and exits.

- **z**

  Suspend telnet. This command enters a sub-shell. The user may return to telnet by exiting the subshell.

- **mode** *type*

  *Type* is either line (for line-by-line mode) or character (for character-at-a-time mode). The remote host is asked for permission to go into the requested mode. If the remote host is capable of entering that mode, the requested mode is entered.

- **status**

  Show the current status of telnet. This includes the peer one is connected to, as well as the current mode.

- **display [** *argument***... ]**

  Displays all, or some, of the set and toggle values (see below).

- **? [** *command* **]**

  Get help. With no arguments, telnet prints a help summary. If a command is specified, telnet prints the help information for just that command.

- **send** *arguments*

  Sends one or more special character sequences to the remote host. The following are the arguments which may be specified (more than one argument may be specified at a time):

  — **escape**

    Sends the current telnet escape character (initially **^]**).

  — **synch**

    Sends the TELNET SYNCH sequence. This sequence causes the remote system to discard all previously typed (but not yet read) input. This sequence is sent as TCP urgent data (and may not work if the remote system is a 4.2 BSD system - if it doesn't work, a lowercase r may be echoed on the terminal).

  — **brk**

    Sends the TELNET BRK (Break) sequence, which may have significance to the remote system.

  — **ip**

    Sends the TELNET IP (Interrupt Process) sequence, which should cause the remote system to abort the currently running process.

  — **ao**

    Sends the TELNET AO (Abort Output) sequence, which should cause the remote system to flush all output from the remote system to the user's terminal.

  — **ayt**

    Sends the TELNET AYT (Are You There) sequence, to which the remote system may choose to respond.

  — **ec**

Sends the TELNET EC (Erase Character) sequence, which should cause the remote system to erase the last character entered.

— **el**

Sends the TELNET EL (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.

— **ga**

Sends the TELNET GA (Go Ahead) sequence, which likely has no significance to the remote system.

— **nop**

Sends the TELNET NOP (No OPeration) sequence.

— **?**

Prints out help information for the send command.

- **set** *argument value*

Set any one of a number of telnet variables to a specific value. The special value off turns off the function associated with the variable. The values of variables may be interrogated with the display command. The variables which may be specified are:

— **echo**

This is the value (initially ^E) which, when in line-by-line mode, toggles between doing local echoing of entered characters (for normal processing), and suppressing echoing of entered characters (for entering, say, a password).

— **escape**

This is the telnet escape character (initially **^[**) which causes entry into telnet command mode (when connected to a remote system).

— **interrupt**

If telnet is in localchars mode (see toggle localchars below) and the interrupt character is typed, a TELNET IP sequence (see send ip above) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal's intr character.

— **quit**

If telnet is in localchars mode (see toggle localchars below) and the quit character is typed, a TELNET BRK sequence (see send brk above) is sent to the remote host. The initial value for the quit character is taken to be the terminal's quit character.

— **flushoutput**

If telnet is in localchars mode (see toggle localchars below) and the flushoutput character is typed, a TELNET AO sequence (see send ao above) is sent to the remote host. The initial value for the flush character is taken to be the terminal's flush character.

— **erase**

If telnet is in localchars mode (see toggle localchars below), and if telnet is operating in character-at-a-time mode, then when this character is typed, a TELNET EC sequence (see send ec above) is sent to the remote system. The initial value for the erase character is taken to be the terminal's erase character.

— **kill**

If telnet is in localchars mode (see toggle localchars below), and if telnet is operating in character-at-a-time mode, then when this character is typed, a TELNET EL sequence (see send el above) is sent to the remote system. The initial value for the kill character is taken to be the terminal's kill character.

— **eof**

If telnet is operating in line-by-line mode, entering this character as the first character on a line causes this character to be sent to the remote system. The initial value of the eof character is taken to be the terminal's eof character.

- **toggle** *arguments***...**

Toggle (between TRUE and FALSE) various flags that control how telnet responds to events. More than one argument may be specified. The state of these flags may be interrogated with the display command. Valid arguments are:

— **localchars**

If this is TRUE, then the flush, interrupt, quit, erase, and kill characters (see set above) are recognized locally, and transformed into (hopefully) appropriate TELNET control sequences (respectively ao, ip, brk, ec, and el; see send above). The initial value for this toggle is TRUE in line-by-line mode, and FALSE in character-at-a-time mode.

— **autoflush**

If autoflush and localchars are both TRUE, then when the ao, intr, or quit characters are recognized (and transformed into TELNET sequences; see set above for details), telnet refuses to display any data on the user's terminal until the remote system acknowledges (via a TELNET Timing Mark option) that it has processed those TELNET sequences. The initial value for this toggle is TRUE if the terminal user had not done an stty noflsh, otherwise FALSE (see stty(1)).

— **autosynch**

If autosynch and localchars are both TRUE, then when either the intr or quit character is typed (see set above for descriptions of the intr and quit characters), the resulting TELNET sequence sent is followed by the TELNET SYNCH sequence. This procedure should cause the remote system to begin throwing away all previously typed input until both of the TELNET sequences have been read and acted upon. The initial value of this toggle is FALSE.

— **crmod**

Toggle carriage return mode. When this mode is enabled, most carriage return characters received from the remote host is mapped into a carriage return followed by a line feed. This mode does not affect those characters typed by the user, only those received from the remote host. This mode is not very useful unless the remote host only sends carriage return, but never line feed. The initial value for this toggle is FALSE.

— **debug**

Toggles socket level debugging (useful only to the superuser ). The initial value for this toggle is FALSE.

— **options**

Toggles the display of some internal telnet protocol processing (having to do with TELNET options). The initial value for this toggle is FALSE.

— **netdata**

Toggles the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.

— **?**

Displays the legal toggle commands.

# Acknowledgements

This section was developed by the University of California, Berkeley.

This LynxOS component is available only as a part of the Lynx TCP/IP package.

# Name

test_node - test the functionality of a node

# Synopsis

**test_node [-debug] [-time** *m***] [-slot** *#***]**

# Description

The test_node command runs a suite of tests on the specified card (or on all cards in the node), unless an initial check shows that the system is not running. Each pass uses TCS to discover line cards, identifies the physical card type and logical card type, and runs the appropriate test. If a card is not operational, the tests are not run on that card. At the end of each pass it reboots the line cards. It can diagnose any card type except the packet line card (PLC) and associated Ethernet and FDDI cards. If it encounters a PLC, it informs you of this and proceeds to the next card. PLCs may be tested with the hardware diagnostics (see the *Installation and Troubleshooting Manual*).

- For low-speed cards, medium-speed cards, and (OC-3) cell line cards, the program sets every port on the card into loopback mode and verifies that the port comes up.
- For a trunk card, Trunk Up-Down protocol packets must make it out each port and back again.
- A low-speed edge port is made a Frame Relay port and the FR LMI protocol is run as an NNI between the NP and the line card; LMI packets must exit the NP, traverse the switch, exit the port, and return, for the port to be declared Up.
- For a medium-speed card or (OC-3) cell line card configured as an edge card, the ports are looped back as UNIs. The test verifies the command path to the CP and the response of the line card to configuration changes; however, the data path out the port and back is not tested because the UNI has no line integrity protocol in this release.

The program reports errors as they occur, and it reports on cards discovered and tested during each pass. At the conclusion of the test sequence, the program prints a summary of test results and exits with an exit status of 0. The program uses SNMP messages to MMA to invoke tests and to retrieve results. MMA sends these SNMP messages to an LCC process which in turn communicates with the specified line card over the switch. If the node is not fully operational, the program reports failure.

# Options

- **-time** *m*

The minimum time to run (in minutes). The entire test package repeats until it finishes a pass after *m* minutes. If *m*=0 the test loops forever (it can be halted with **^C**). The default is to perform one pass. The switch may be abbreviated to -t.

- **-slot** *#*

  The card number *#* of a slot to test. The default is to test all cards. The switch may be abbreviated to -s.

- **-debug**

  Extremely verbose output used for debugging the program itself. The switch may be abbreviated to -d.

# Name

touch - change the modify date of files

# Synopsis

**touch [-cf]** *file* **...**

# Description

The touch command sets the modification date of a file to the current date. If the file exists, the date is changed by reading a character from it and then writing it back. If the file does not exist, it is created.

# Options

- **-c**

If the file does not exist, do not create it.

- **-f**

  Perform a touch even if read and write permissions are denied on the file.

# See Also

Utility Programs — ls, mkdir

System calls — stat()

# Name

vi - visual text editor

# Synopsis

**vi [ file ... ]**

**vi -t tag**

## Description

vi is a visual text editor, a powerful and convenient tool for the creation, modification, and examination of text files. If vi is invoked with no arguments, it initializes an editing buffer and allows a new (as yet unnamed) file to be created. If given one or more file names, vi allows each file to be examined or modified, one after the other. Any of the files may be nonexistent upon invocation; this case is similar to the case in which vi is started with no names, except that the new file is given the name chosen upon entry.

## Option

- **-t** *tag*

  vi looks in the file tags in the current directory for an entry corresponding to *tag*. If found, vi opens the file named in tags and goes to the position specified. Tag files are usually created with the ctags utility.

## See Also

Utility Programs — ctags

Documents — *Vi User's Guide*

**6 x 6 x**

# Vi Editor User's Guide

This chapter contains a user's guide to the **vi** editor, adapted for LightStream users from the LynxOS documentation set.

## Introduction

The **vi** program is a full-screen text editor for creating and modifying files. Although it was designed for use as a software development tool, **vi** is suitable for a wide variety of editing tasks

The **vi** program determines the type of terminal on which it is being run by examining the TERM environment variable and the /etc/termcap file (or the TERMCAP environment variable). Any terminal defined in the /etc/termcap file can be used, as long as it supports at least cursor positioning and screen erase. On a LightStream node, TERM is set to vt100 in the .profile command file at login time. To change the value, type TERM=*value* at the bash prompt. For example, to set it to sun, enter the following command (**tset** is not needed):

```
bash$ TERM=sun
bash$
```

## Editing Basics

If **vi** is started with no arguments, it creates a new, empty file which can be given a name and written to disk at any time. It may be started with the name of a new or existing file on the command line. (See the section "Editor Initialization" for more details.)

The editor marks lines on the screen which are not part of the file with a single tilde (~). The editor always forces at least one line to exist.

The editor is always in *command* mode or *insert* mode (with some variants of insert mode). Command mode is the initial state of the editor, and is the mode from which nearly all editing commands are issued. Insert mode is used to add or modify text. Several commands switch the editor from command mode to insert mode, with side effects described below.

# Getting Started

Because of the way **vi** is designed, one must know a large number of commands before editing becomes really convenient. However, a few simple commands are enough for rudimentary file operations, and are certainly enough for a careful typist to create new documents.

The first thing to learn is how to get to insert mode, since this is the normal way to add text to a file. The editor starts in command mode, and the entire keyboard is "hot": any key pressed results in immediate action of some kind. The i key puts the editor in insert mode. Unfortunately, there is no indication of this change, visual or otherwise. In insert mode, all printable characters that are typed are inserted into the file before the current character. The return key terminates a line and moves to the next line on the screen. The backspace key (**^H**, control-H) can be used to back up and fix typing errors, but only within the current line. Similarly, **^W** erases over the most recently typed word, and **^U** erases up to the beginning of the line.

The **[Esc]** key, or control - [, returns from insert mode to command mode. In command mode, the h key moves one character left, l moves one character right, k moves one line up, and j moves one line down. (These keys form the line hjkl on the keyboard.) The editor sounds the bell if it is asked to move to a meaningless position, such as before the beginning of a line or after the end of the file.

The **x** key deletes individual characters. To delete a line, type d twice in succession. If something is deleted accidentally, recover it by typing u. Type i to re-enter insert mode at any time.

There are two ways to terminate the editing session. Type z twice to save the current file and leave the editor. Type the string :q! plus carriage return to abort the editing session without saving changes made to the file.

# Terminology

The following terms are used in the descriptions of commands:

**Table 6-1 Terms Used in Command Descriptions**

| Terms Used in Command Descriptions | |
|---|---|
| text | Any series of ASCII characters. |
| buffer | A unit of memory used to hold text. |
| file buffer | The buffer that holds the file being edited. |
| white space | Tabs and spaces. |
| word | An unbroken string of (1) letters, digits, and underscores ("_"), or (2) characters other than letters, digits, underscores, and white space, depending upon context. |
| big word | An unbroken string of characters other than white space. |
| unnamed buffer | A buffer containing the last text yanked or deleted. |
| window | The part of the file that is visible on the terminal. |

| Terms Used in Command Descriptions | |
|---|---|
| line | A string of characters followed by a newline character. A “line” of text may take up more than one row on the screen. |
| current character | The character beneath the cursor. |
| current line | The line containing the cursor. |
| ^*x* | Control character *x*, e.g. ^**B** means control-B (ASCII 2). |
| return | **^M** or the **[Return]** key. |
| command mode | The normal operating mode of the editor. Each key typed is taken as a command. |
| insert mode | The mode used for manually adding text to a file. Characters that are typed while in insert mode are inserted into the text. |
| replace mode | Like insert mode, except that characters typed replace characters in the buffer. |
| char | Any ASCII character. |
| named buffer | One of 26 buffers with names “a” through “z”. |
| mark name | A lowercase letter that names one of 26 marks that can be used to save a position in the file buffer. |

# vi Regular Expressions

Regular expressions are usually used for searching. Most characters match themselves in a search request, but the following characters have special meanings:

**Table 6-2 Regular Expressions Used in vi**

| Regular Expressions Used in vi | |
|---|---|
| | Matches any single character. |
| **[***chars***]** | Matches any *char* listed in the brackets. **[***x*-*y***]** is a range of characters, where the ASCII value of *x* precedes that of *y*. |
| **[^***chars***]** | Matches any *char* not listed in the brackets. |
| ^ | Matches the beginning of a line when used as the first character in a regular expression. |
| $ | Matches the end of a line when used as the last character in a regular expression. |
| * | Matches zero or more instances of the preceding item, which may be a normal *char* or a 1-character “wildcard” such as **.** or **[***chars***]**. |
| + | Matches one or more instances of the preceding item. |
| ? | Matches zero or one instances of the preceding item. |
| \< | Matches the beginning of a word |
| \> | Matches the end of a word. |

This feature may be turned off with the command **set nomagic**, as described in Table 6-12. To search for one of these special characters, precede it by a backslash to prevent special interpretation. When special searching characters make several matches possible in a single line, **vi** matches the longest string possible. This becomes important for search-replace commands.

**Note** Parentheses and angle brackets are interpreted literally, and need not be preceded with backslashes.

# Position Movement Commands

There are two kinds of movement commands. Position movement commands move the current position in the file based on file contents. These commands are shown below. Note that uppercase and lowercase letters are distinct. There are commands for nearly every editing situation, but only a few are needed by beginners for simple editing.

With most position movement commands, optionally type an integer *n* before the command (with no space) to repeat the movement *n* times. The default is 1.

**Table 6-3 Position Movement Commands**

| Position Movement Commands | |
|---|---|
| *n***h** (or *n*^**H**) | Left *n* characters (1 by default). Within current line only. |
| *n***j** (or *n*^**J**, or *n*^**N**) | Down *n* lines (1 by default), in the same column if possible. |
| *n***k** (or *n*^**K**) | Up *n* lines (1 by default), in the same column if possible. |
| *n***l** (or *n*^**L**, or *nspace*) | Right *n* characters (1 by default). |
| ^**I** | Forward to the next tab stop. |
| *n***w** | Forward to the beginning of the *n*th following *word*. |
| *n***W** | Forward to the beginning of the *n*th following *big word*. |
| *n***b** | Back to the beginning of the *n*th previous *word*. |
| *n***B** | Back to the beginning of the *n*th previous *big word*. |
| *n***e** | Forward to the end of the *n*th following *word*. |
| *n***E** | Forward to the end of the *n*th following *big word*. |
| *n***G** | The *n*th line of the file (default, last line). |
| **0** | The first character of the current line. |
| **_** (or ^) | The first non-whitespace character on the current line. |
| **$** | The end of the current line. |
| **%** | The matching parenthesis, bracket, or brace. |
| *'mark* | The beginning of the line previously marked with *mark* (a single lowercase letter). |
| *''* | If *mark* is **'**, move to the beginning of the line that was current just before the last absolute movement command such as **G**, **%**, or **'** itself. |
| **`***mark* | The position (line and column) previously marked with *mark* (a single lowercase letter). |
| **``** | If *mark* is **`**, move to the position that was current just before the last absolute movement command. |
| *n*+ | The first non-whitespace character on the *n*th line below. |
| *n*- | The first non-whitespace character on the *n*th line above. |

| Position Movement Commands | |
|---|---|
| / | Search forward. Prompts on the last line of the screen for a search pattern (default = last pattern searched for). |
| **?** | Search backward. Prompts on the last line of the screen for a search pattern (default = last pattern searched for). |
| **n** | Search forward for last pattern searched for. |
| **N** | Search backward for last pattern searched for. |
| **f***char* | Next instance of *char* on current line. |
| **t***char* | Character preceding next instance of *char* on current line. |
| **F***char* | Previous instance of *char* on current line. |
| **T***char* | Character following previous instance of *char* on current line. |
| **;** | Repeat last **t** or **f** command. |
| **,** | Repeat last **T** or **F** command. |
| *n***H** | Beginning of *n*th line from top of text (default = 1, home). |
| *n***L** | Beginning of line *n*th row from bottom of text (default = 1). |
| *n*^**M** | First non-whitespace character on the *n*th line below (default = carriage return). |

All the position movement commands can be used as operands to various change, delete, and substitute commands described farther on.

# Secondary Movement Commands

Use secondary movement commands to move quickly through the file or to reposition the window in the file. Secondary movement commands cannot be used as operands to commands that change text.

**Table 6-4 Secondary Movement Commands**

| Secondary Movement Commands | |
|---|---|
| ^**B** | Move a full window backward. |
| ^**D** | Move half a window forward. |
| ^**E** | Move the window down one line. The cursor moves only if the current line scrolls off the screen. |
| ^**F** | Move a full window forward. |
| ^**U** | Move half a window backward. |
| ^**Y** | Move the window up one line. The cursor moves only if the current line scrolls off the screen. |
| **z.** | Scroll the current line to the middle of the window. |
| **z-** | Scroll the current line to the bottom of the window. |
| **z***return* | Scroll the current line to the top of the window. |

# Text-Changing Commands

There are two kinds of commands that change text, those that are used with arguments and operands (movement commands), and those that are used alone.

The effect of any command that changes text can be repeated with the **.** command, or undone with the **u** or **U** command. Consequently, the sequence of entering insert mode, adding text, and returning to command mode can be repeated several times, or undone if the added text is unsatisfactory.

Several of the text-changing commands put a copy of the changed text into the unnamed buffer. These commands are **c**, **d**, **s**, **x**, and **X**. Use **p** or **P** to retrieve the changed text.

## Text-Changing Commands With Operands

The most important of the text-changing commands in **vi** operate on a variable region of text determined by an argument or operand of the command. For example, with a single command one may change the next three words, or delete lines up to the first instance of some pattern, or horizontally shift lines between the current position and some marked line.

Each of the commands that take operators or arguments is normally followed by one of the primary movement commands. The current cursor position defines one end of the region to be changed, and the movement command defines the other end. Alternatively, if a command is doubled, it affects the entire current line; for example, **dd** deletes the current line. If a repeat count *n* is included, it can be typed before the change command or before the movement command.

**Table 6-5 Commands That Change a Region Delimited by a Movement Command**

| Commands That Change a Region Delimited by a Movement Command | |
|---|---|
| **c** | Change the region of text delimited by the movement operand. If the end of the text region is within the current line, **vi** marks it with a **$** mark (dollar sign), enters replace mode until the cursor reaches the **$** mark, and then enters insert mode for additional typing. Otherwise, **vi** deletes text from the current position through the end of the region and enters insert mode. |
| **d** | Delete the region of text. Equivalent to using c and immediately pressing **[Esc]** to leave insert mode. |
| **y** | Copy text unchanged from the region to the unnamed buffer. |
| < | Shift each line of the region left one tab stop. |
| > | Shift each line of the region right one tab stop. |
| **!** | Filter the region through a LynxOS utility or other program. Prompts for a command. Standard output of the command replaces the text in the region. |

## Text-Changing Commands Without Operands

Other text-changing commands do not require operands, because they operate on predefined regions of text, such as individual characters or lines, and enter insert mode in various ways. A repeat count *n* may be given with many of these commands (default = 1).

**Table 6-6 Commands That Change a Predefined Text Object**

| Commands That Change a Predefined Text Object | |
|---|---|
| **i** | Enter insert mode before the current character. |
| **I** | Move to the first non-whitespace character of the current line and enter insert mode. |
| **a** | Enter insert mode after the current character. |
| **A** | Move to the last character of the current line and enter insert mode after that character. |
| **o** | Add a blank line below the current line, move to the new line, and enter insert mode. |
| **O** | Add a blank line above the current line, move to the new line, and enter insert mode. |

| Commands That Change a Predefined Text Object | |
|---|---|
| *n***x** | Delete *n* characters to the right, starting with current character. |
| *n***X** | Delete *n* characters to the left of current character. |
| **J** | Join the next line with the current line. |
| *n***p** | Insert text from the unnamed buffer *n* times after the current position. Insert after the current line if the buffer contains lines of text. |
| *n***P** | Insert text from the unnamed buffer *n* times before the current position. Insert before the current line if the buffer contains lines of text. |
| **R** | Enter replace mode, a variant of insert mode. Typing overwrites characters on the current line. As soon as the current line is completely overwritten, enter insert mode. |
| **r***char* | Replace the current character with *char*. |
| **s** | Change the current character (same as **c1**). |
| **C** | Change the text in the current line (same as **c$**). |
| **D** | Delete the remainder of the current line (same as **d$**). |
| **Y** | Copy the remainder of the current line into the unnamed buffer (same as **y$**). |
| **S** | Change the current line (same as **cc**). |
| **u** | Undo the effects of the last text-changing command. |
| **U** | Multi-level undoing through the 9 most recent changes. (By contrast, repeating **u** has no net effect on the file.) |
| **&** | Repeat last search/replace command (see Colon Commands). |
| ~ | Change case (uppercase or lowercase) of the current character, if it is alphabetic, and move right one character. |
| **.** | Repeat the effect of the last text-changing command. |

# Miscellaneous Commands

The following commands are useful and important but defy ready categorization:

**Table 6-7 Miscellaneous Commands**

| Miscellaneous Commands | |
|---|---|
| **m***mark* | Mark the current line with *mark*, a single lowercase letter, for future reference by movement and change commands. |
| ^**R** | Redraw the screen. |
| ^**G** | Display information about the current position and file buffer. |
| ^^ | Toggle between the current file buffer and the last file buffer edited. |
| **"***letter* | Use the buffer named *letter* for the following command. **P** or **p** inserts from *letter* instead of from the unnamed buffer, and commands that write into the unnamed buffer write into *letter* as well. |
| **@***letter* | Perform commands found in the buffer named *letter* as if they were typed on the keyboard. To put commands into a buffer, create a new line in the main file and then delete it into the buffer. The **.** command repeats the effects of an invocation of buffered commands; the **u** and **U** commands repeat the effects of individual commands in the buffer. |
| ^**]** | Interpret characters from the current position to the end of the word as a tag name, and go to that tag position. If this requires editing a new file, the current file must first be saved or the autowrite flag must be set. (See the **ta** command description in Table 6-10.) |

| Miscellaneous Commands | |
|---|---|
| **ZZ** | Save the current file and leave **vi**. (See also **:xit** in Table 6-10.) |
| **^C** | Abort a search in progress or a colon command being entered. |
| **^Z** | Put the current editing session in the background and return to the invoking shell. To resume editing, use the appropriate shell command to bring **vi** back into the foreground. |

# Colon Commands

The **:** command accesses a family of commands which cannot easily be expressed by one or two keystrokes.

- When **:** is typed, **vi** prompts for a colon command.
- Colon commands may be abbreviated: one need type only enough characters to distinguish the intended command from other colon commands, then press **[Return]** or **[Esc]** to terminate the command.
- Several colon commands can be put on a single line separated by | characters, in the manner of a shell pipeline.

## Colon Commands with a Range of Lines

The following colon commands operate on a range of lines within a file:

**Table 6-8 Colon Commands**

| Colon Commands | |
|---|---|
| *range* **s**/*pat*/*repl*/**g** | The **s** command substitutes the replacement text *repl* in place of the pattern text *pat* on each line included in the *range*. If *range* is omitted, the default is the current line. If the optional **g** is included at the end of the command, each occurrence of *pat* is replaced, otherwise only the first instance on each line is affected.<br>The pattern *pat* may be any regular expression (see Table 6-2). If the replacement string *repl* contains the character **&**, the current instance of *pat* is substituted for it. If the replacement string *repl* contains the sequence \n, where *n* is a single digit from 1 to 9, that sequence is replaced by the *n*th portion of *pat* that is bracketed by **\(** and **\)**.<br>Examples:<br>Change every instance of cat to dog:<br>**:%s/cat/dog/g**<br>Remove all leading and training spaces in a file:<br>**:%s/^ *\(.*[^ ]\) *$.\1**<br>Note that the pattern grouped in **\(** and **\)** explicitly specifies that the pattern end with a non-space. This is because **vi** always matches the longest patterns and sub-patterns.<br>Parenthesize the first string of capital letters on the current line:<br>**:s/[A-Z]+/(&)/** |
| *range* **w** *file*<br>*range* **w!** *file*<br>*range* **w>>** *file* | Write the lines in *range* to *file*. If *range* is omitted, write the entire file; in the absence of *file*, use the current file name. Use **w!** to overwrite an existing file. Use **w>>** to append to the end of an existing file. |

The *range* parameter of the above commands consists of one or two of the following line specifiers:

**Table 6-9 Line Specifier Strings Used to Specify Range of Colon Commands**

| Line Specifier Strings Used to Specify Range of Colon Commands | |
|---|---|
| **.** | The current line. |
| **$** | The last line of the file. |
| *n* | Line number *n* of the file (a decimal integer). |
| **'***mark* | Line previously marked by the 1-character *mark* with the **m** command. |

If *range* consists of only one line specifier, it selects just that one line. If *range* consists of two line specifiers separated by a comma, it selects all lines from the first to the second, inclusive, regardless of the order in which the two line specifiers are entered. If no *range* is given, the default may be the current line or the entire file, depending on the command. The range specifier % is an abbreviation for 1,$, selecting the entire file.

## Colon Commands That Specify Files

The colon commands that follow determine the file to be edited:

**Table 6-10 Colon Commands That Determine the File Being Edited**

| Colon Commands That Determine the File Being Edited | |
|---|---|
| **edit** *file* | Start editing another file. If *file* is not given, the default is to reload the version of the current file found on disk (the last saved version, or the original if it has not been saved). If *file* is #, then **vi** switches to the alternate file, which is either the most recent file edited in this **vi** session, or the file most recently named in a colon command, even if that command failed. If the current file has not been saved, **vi** does not switch files unless the autowrite flag is set. It may be forced to switch by typing **edit!** *file*, but all changes to the current file are then lost. |
| **next** | Edit the next file in the parameter list that was given when **vi** was invoked. If the current file has not been saved, **vi** does not switch files unless the autowrite flag is set. It may be forced to switch by typing **next!**, but all changes to the current file are then lost. |
| **quit** | Exit **vi**. If the current file has not been saved, **vi** does not terminate unless the autowrite flag is set. It may be forced to exit by typing **quit!**, but all changes to the current file are then lost. |
| **read** *file* | Read the contents of *file* into the file buffer after the current line. |
| **rewind** | Change to the first parameter in the parameter list given when **vi** was invoked. The other parameters in the list can then be reexamined with the next command. If the current file has not been saved, **vi** does not switch files unless the autowrite flag is set. It may be forced to switch by typing **rewind!**, but all changes to the current file are then lost. |
| **ta** *tag* | Look up the string *tag* in a file named tags in the current working directory. Lines in this file are of the following form:<br>*tag filename pattern*<br>When it finds *tag*, **vi** changes to the file *filename* (if it is different from the current file) and moves to the first instance of *pattern* in that file. The tags file is usually constructed automatically, by the **ctags** program if the files contain C source code, or by some other program for other types of files. If the current file has not been saved, **vi** does not switch files unless the autowrite flag is set. It may be forced to switch by typing **ta!** *tag*, but all changes to the current file are then lost. |
| **xit** | Exit **vi** after saving the current file buffer. (See also **ZZ** Table 6-7.) |

## Commands That Tailor the Editing Environment

Several colon commands can be used to create keyboard shortcuts. The **:set** command can be used to manipulate the **vi** editor parameters described in Table 6-12. These commands are as follows:

**Table 6-11 Commands That Tailor the Editing Environment**

| Commands That Tailor the Editing Environment | |
|---|---|
| **map** *key string* | Assign user-definable strings to particular characters. If *key* is typed in command mode, vi behaves as if *string* had been typed instead (but see below for map!). |
| **map!** *key string* | If *key* is typed in insert mode, vi behaves as if *string* had been typed instead.<br>The same *key* can be used twice, once for insert mode and once for command mode. |
| **unmap** *key* | Remove any mapping established for *key* in command mode. |
| **unmap!** *key* | Remove any mapping established for *key* in insert mode. |
| **set** *flag* | Set *flag* to TRUE. |
| **set no***flag* | Set *flag* to FALSE. |
| **set** *var=value* | Set *var* to the numeric value *value*. |

The following **vi** editor parameters may be manipulated by **:set** commands.

**Table 6-12 Editor Parameters That May be Set With the set Command**

| Editor Parameters That May be Set With the set Command | |
|---|---|
| **autoindent** | If set, when text is added vi indents each new line by the same amount as the previous line. |
| **autowrite** | If set, vi writes the file to disk automatically when it switches between files or when it is stopped with ^Z. |
| **ignorecase** | If set, search commands ignore case. |
| **list** | If set, tabs in the file are made visible on the screen as ^I. |
| **magic** | If *not* set (**set nomagic**), special characters described in Table 6-2 need not be quoted with a backslash. |
| **tabstop** | Width in spaces of a tab stop; used also for shifting text. |
| **wrapscan** | If set, searches wrap around from one end of the file to the other. |

# Editor Initialization

When **vi** starts up, it looks for the variable VIINIT in the user's shell environment. If VIINIT is not set, **vi** looks for a file called .virc in the user's home directory. What it looks for is a list of colon commands, delimited by newlines, which it executes.

After executing commands found in VIINIT or $HOME/.virc, then **vi** looks for a .virc file in the current working directory. Because the commands in a local .virc take effect after those described above, the local file may be used to set parameters for editing a special set of files grouped in one directory.

If **vi** is started with the -t option, it uses a tag file created by **ctags** or some other method to start and load the file containing a requested tag. If a positioning command (see Table 6-3) is given before any tag or file names, **vi** starts at the specified position rather than at the beginning of the file.

# Command Summary

All of the **vi** commands described above are given in ASCII order in the list that follows. Unused characters are also listed.

**Table 6-13 Command-Mode Command Summary**

| | |
|---|---|
| **^A** | Not used. |
| **^B** | Move a full window backward. |
| **^C** | Abort a search in progress or a colon command being entered. |
| **^D** | Move half a window forward. |
| **^E** | Move the window down one line. The cursor moves only if the current line scrolls off the screen. |
| **^F** | Move a full window forward. |
| **^G** | Display information about the current position and file buffer. |
| *n***^H** | Left *n* characters (1 by default). Within current line only. |
| **^I** | Forward to the next tab stop. |
| *n***^J** | Down *n* lines (1 by default), in the same column if possible. |
| *n***^K** | Up *n* lines (1 by default), in the same column if possible. |
| *n***^L** | Right *n* characters (1 by default). |
| *n***^M** | First non-whitespace character on the *n*th line below (default = carriage return). |
| *n***^N** | Down *n* lines (1 by default), in the same column if possible. |
| **^O** | Not used. |
| **^P** | Not used. |
| **^Q** | Not used (XON character for flow control). |
| **^R** | Redraw the screen. |
| **^S** | Not used (XOFF character for flow control). |
| **^T** | Not used. |
| **^U** | Move half a window backward. |
| **^V** | Not used. |
| **^W** | Not used. |
| **^X** | Not used. |
| **^Y** | Move the window up one line. The cursor moves only if the current line scrolls off the screen. |
| **^Z** | Put the current editing session in the background and return to the invoking shell. To resume editing, use the appropriate shell command to bring vi back into the foreground. |
| **^[** | Not used in command mode (exits insert mode). |
| ^\ | Not used. |
| **^]** | Interpret characters from the current position to the end of the word as a tag name, and go to that tag position. If this requires editing a new file, the current file must first be saved or the autowrite flag must be set. (See the ta command below under Colon Commands.) |
| ^^ | Toggle between the current file buffer and the last file buffer edited. |
| ^_ | Not used. |
| *nspace* | Right *n* characters (1 by default). |

| | |
|---|---|
| **!** | Filter the region through a LynxOS utility or other program. Prompts for a command. Standard output of the command replaces the text in the region. |
| **"***letter* | Use the buffer named *letter* for the following command. P or p inserts from *letter* instead of from the unnamed buffer, and commands that write into the unnamed buffer write into *letter* as well. |
| **#** | Not used. |
| **$** | The end of the current line. |
| **%** | The matching parenthesis, bracket, or brace. |
| **&** | Repeat last search/replace command (see Colon Commands). |
| **'***mark* | The beginning of the line previously marked with *mark* (a single lowercase letter). |
| **''** | If *mark* is **'**, move to the beginning of the line that was current just before the last absolute movement command such as G, %, or ' itself. |
| **(** | Not used. |
| **)** | Not used. |
| ***** | Not used. |
| *n*+ | The first non-whitespace character on the *n*th line below. |
| **,** | Repeat last T or F command. |
| *n*- | The first non-whitespace character on the *n*th line above. |
| | Repeat the effect of the last text-changing command. |
| / | Search forward. Prompts on the last line of the screen for a search pattern (default = last pattern searched for). |
| **0** | The first character of the current line. |
| **1 ... 9** | Not used, except to specify repeat counts to other commands. |
| **:** | Used to introduce a colon command. |
| **;** | Repeat last t or f command. |
| < | Shift each line of the region left one tab stop. |
| **=** | Not used. |
| > | Shift each line of the region right one tab stop. |
| **?** | Search backward. Prompts on the last line of the screen for a search pattern (default = last pattern searched for). |
| **@***letter* | Perform commands found in the buffer named *letter* as if they were typed on the keyboard. Put commands into a buffer by creating a new line in the main file and then deleting it into the buffer. Use the . command to repeat the effects of an invocation of buffered commands; u and U commands only repeat the effects of individual commands in the buffer. |
| **A** | Move to the last character of the current line and enter insert mode after that character. |
| *n***B** | Back to the beginning of the *n*th previous *big word*. |
| **C** | Change text in the current line (same as c$). |
| **D** | Delete remainder of the current line (same as d$). |
| *n***E** | Forward to the end of the *n*th following *big word*. |
| **F***char* | Previous instance of *char* on current line. |
| *n***G** | The *n*th line of the file (default, last line). |
| *n***H** | Beginning of *n*th line from top of text (default = 1, home). |
| **I** | Move to the first non-whitespace character of the current line and enter insert mode. |
| **J** | Join the next line with the current line. |

| | |
|---|---|
| **K** | Not used. |
| **n**L | Beginning of line *n*th row from bottom of text (default = 1). |
| **M** | Not used. |
| **N** | Search backward for last pattern searched for. |
| **O** | Add a blank line above the current line, move to the new line, and enter insert mode. |
| *n***P** | Insert text from the unnamed buffer *n* times before the current position. Insert before the current line if the buffer contains lines of text. |
| **Q** | Not used. |
| **R** | Enter replace mode, a variant of insert mode. Typing overwrites characters on the current line. As soon as the current line is completely overwritten, enter insert mode. |
| **S** | Change the current line (same as cc). |
| **T***char* | Character following previous instance of *char* on current line. |
| **U** | Multi-level undoing through the 9 most recent changes. (By contrast, repeating u has no net effect on the file.) |
| **V** | Not used. |
| *n***W** | Forward to the beginning of the *n*th following *big word*. |
| *n***X** | Delete *n* characters to the left of current character. |
| **Y** | Copy remainder of the current line into the unnamed buffer (same as y$). |
| **ZZ** | Save the current file and leave vi. |
| **[** | Not used. |
| \ | Not used. |
| **]** | Not used. |
| ^ | The first non-whitespace character on the current line. |
| _ | The first non-whitespace character on the current line. |
| **‘***mark* | The position (line and column) previously marked with *mark* (a single lowercase letter). |
| **‘‘** | If *mark* is **‘**, move to the position that was current just before the last absolute movement command. |
| **a** | Enter insert mode after the current character. |
| *n***b** | Back to the beginning of the *n*th previous *word*. |
| **c** | Change the region of text delimited by the movement operand. If the end of the text region is within the current line, vi marks it with a $ mark (dollar sign), enters replace mode until the cursor reaches the $ mark, and then enters insert mode for additional typing. Otherwise, vi deletes text from the current position through the end of the region and enters insert mode. |
| **d** | Delete the region of text. Equivalent to using c and immediately pressing [Esc] to leave insert mode. |
| *n***e** | Forward to the end of the *n*th following *word*. |
| **f***char* | Next instance of *char* on current line. |
| **g** | Not used. |
| *n***h** | Left *n* characters (1 by default). Within current line only. |
| **i** | Enter insert mode before the current character. |
| *n***j** | Down *n* lines (1 by default), in the same column if possible. |
| *n***k** | Up *n* lines (1 by default), in the same column if possible. |
| *n***l** | Right *n* characters (1 by default). |

| | |
|---|---|
| **m***mark* | Mark the current line with *mark*, a single lowercase letter, for future reference by movement and change commands. |
| **n** | Search forward for last pattern searched for. |
| **o** | Add a blank line below the current line, move to the new line, and enter insert mode. |
| *n***p** | Insert text from the unnamed buffer *n* times after the current position. Insert after the current line if the buffer contains lines of text. |
| **q** | Not used. |
| **r***char* | Replace the current character with *char*. |
| **s** | Change the current character (same as c1). |
| **t***char* | Character preceding next instance of *char* on current line. |
| **u** | Undo the effects of the last text-changing command. |
| **v** | Not used. |
| *n***w** | Forward to the beginning of the *n*th following *word*. |
| *n***x** | Delete *n* characters to the right, starting with current character. |
| **y** | Copy text unchanged from the region to the unnamed buffer. |
| **z***return* | Scroll the current line to the top of the window. |
| **z-** | Scroll the current line to the bottom of the window. |
| **z.** | Scroll the current line to the middle of the window. |
| **{** | Not used. |
| **\|** | Not used. |
| **}** | Not used. |
| **~** | Change case (uppercase or lowercase) of the current character, if it is alphabetic, and move right one character. |

# Notes on LynxOS vi

The following list summarizes the salient differences between the version of **vi** running under LynxOS and other popular versions of the **vi** editor. Many of the differences are enhancements. Differences that reflect bugs found in other versions are not listed here.

- **U** undoes the last text change up to 9 changes. In other versions, **U** undoes all changes made to the current line since it became the current line.
- After an undo command that causes a change in file position, LynxOS **vi** remembers the previous position so that `''` returns.
- The unnamed buffer is not deleted after the **edit** colon command so that text can be more easily transferred from one file to another.
- The **p** and **P** commands accept a repeat count.
- Backspace can be used to delete whitespace inserted automatically in autoindent mode. Other **vi** versions only recognize ^**D** for this purpose.
- The + and **?** regular expression operators are available in LynxOS **vi**.
- The **Y** command is defined as **y$** to be consistent with the **C** and **D** commands. Other versions of **vi** treat **Y** as if it were **yy**.
- There is no “open” mode, a terminal must support cursor movement to be used with LynxOS **vi**.

- All shift commands can be repeated with the **.** command.
- The **n** command always searches forward, and the **N** command always searches backward. In other versions of **vi**, the **n** command searches in the most recent search direction and **N** searches in the opposite of the most recent direction.
- LynxOS **vi** reads startup commands from the VIINIT environment variable instead of the EXINIT variable.

**7 x A x**

# BASH Shell Reference

This appendix contains the manual page for the bash shell, produced here for LightStream users. The **shell** command in CLI accesses the bash shell so that you can execute LynxOS commands. You can also execute LynxOS commands from the bash shell when you log in as superuser (root).

Numerous commands are built in to the shell (see the section “Shell Builtin Commands”). Of particular interest are the commands **cd**, **kill**, and **pwd**. The **help** command displays help information about all the shell builtin commands.

## Name

bash - GNU Bourne-Again SHell

### Synopsis

**bash [***options***] [***file***]**

## Copyright



### Description

Bash is an sh-compatible command language interpreter that executes commands read from the standard input or from a file. Bash also incorporates useful features from the *Korn* and *C* shells (ksh and csh).

Bash is ultimately intended to be a faithful implementation of the IEEE Posix Shell and Tools specification (IEEE Working Group 1003.2).

### Options

In addition to the single-character shell options documented in the description of the set builtin command, bash interprets the following flags when it is invoked:

- **-c** *string*

  If the -c flag is present, then commands are read from *string*.

- **-i**

If the -i flag is present, the shell is *interactive*.

- **-s**

  If the -s flag is present, or if no arguments remain after option processing, then commands are read from the standard input. This option allows the positional parameters to be set when invoking an interactive shell.

- **-**

  A single - signals the end of options and disables further option processing. Any arguments after the - are treated as filenames and arguments. An argument of -- is equivalent to an argument of -.

Bash also interprets a number of multi-character options. To be recognized, these options must appear on the command line before the single-character options.

- **-norc**

  Do not load the personal initialization file ~/.bashrc if the shell is interactive. This is the default if the shell name is sh.

- **-noprofile**

  Do not read either /etc/profile or ~/.bash_profile. By default, bash normally reads these files when it is invoked as a login shell.

- **-rcfile file**

  Execute commands from file instead of the standard personal initialization file ~/.bashrc, if the shell is interactive.

- **-version**

  Show the version number of this instance of bash when starting.

- **-quiet**

  Do not be verbose when starting up (do not show the shell version or any other information).

- **-login**

  Make bash act as if it had been invoked by login(1).

- **-nobraceexpansion**

  Do not perform curly brace expansion as csh does.

- **-nolineediting**

  Do not use the GNU readline library to read command lines if interactive.

## Arguments

If arguments remain after option processing, and neither the -c nor the -s option has been supplied, the first argument is assumed to be the name of a file containing shell commands. If bash is invoked in this fashion, $0 is set to the name of the file, and the positional parameters are set to the remaining arguments. Bash reads and executes commands from this file, then exits.

## Definitions

- blank

  A space or tab.

- word

  A sequence of characters considered as a single unit by the shell. Also known as a token.

- name

  A *word* consisting only of alphanumeric characters and underscores, and beginning with an alphabetic character or an underscore. Also referred to as an identifier.

- metacharacter

  A character that, when unquoted, separates words. One of the following:

  ```
  | & ; ( ) < > <space> <tab>
  ```

- control operator

  A *token* that performs a control function. It is one of the following symbols:

  ```
  || & && ; ;; ( ) | <newline>
  ```

## Reserved Words

*Reserved words* are words that have a special meaning to the shell. The following words are recognized as reserved when unquoted and either the first word of a simple command (see Shell Grammar below) or the third word of a case or for command:

```
! casedo done elif else esac fi for function if in then until while { }
```

## Shell Grammar

### Simple Commands

A *simple command* is a sequence of optional variable assignments followed by *blank*-separated words and redirections, and terminated by a *control operator*. The first word specifies the command to be executed. The remaining words are passed as arguments to the invoked command.

The return value of a *simple command* is its exit status, or 128+*n* if the command is terminated by signal *n*.

### Pipelines

A *pipeline* is a sequence of one or more commands separated by the character |. The format for a pipeline is as follows:

**[ ! ]** *command* **[** | *command2* **... ]**

The standard output of *command* is connected to the standard input of *command2*. This connection is performed before any redirections specified by the command (see Redirection below).

If the reserved word ! precedes a pipeline, the exit status of that pipeline is the logical NOT of the exit status of the last command. Otherwise, the status of the pipeline is the exit status of the last command. The shell waits for all commands in the pipeline to terminate before returning a value.

Each command in a pipeline is executed as a separate process (i.e. in a subshell).

## Lists

A list is a sequence of one or more pipelines separated by one of the operators ;, &, &&, or ||, and optionally terminated by one of ;, &, or <*newline*>.

Of these list operators, && has highest precedence, || has the next highest precedence, followed by ; and &, which have equal precedence.

If a command is terminated by the control operator &, the shell executes the command in the background in a subshell. The shell does not wait for the command to finish. Commands separated by a ; are executed sequentially; the shell waits for each command to terminate in turn.

The control operators && and || denote AND lists and OR lists, respectively. An AND list has the following form:

*command* **&&** *command2*

Here, *command2* is executed if, and only if, *command* returns an exit status of zero. An OR list has the following form:

*command* **||** *command2*

Here, *command2* is executed if and only if *command* returns a nonzero exit status.

## Compound Commands

A *compound command* is one of the following:

- **(***list***)**

  *list* is executed in a subshell. Variable assignments and builtin commands that affect the shell's environment do not remain in effect after the command completes.

- **{** *list***; }**

  *list* is simply executed in the current shell environment. This is known as a *group command*.

- **for name [ in** *word***; ] do** *list* **; done**

  The list of words following in is expanded, generating a list of items. The variable *name* is set to each element of this list in turn, and *list* is executed each time. If in *word* is omitted, the for command executes *list* once for each positional parameter that is set (see Parameters below). The exit status is the exit status of the last command, or zero if no commands were executed.

- **case** *word* **in [** *pattern* **[** | *pattern* **] ... )** *list* **;; ] ... esac**

  A case command first expands *word*, and tries to match it against each *pattern* in turn. When a match is found, the corresponding *list* is executed. After the first match, no subsequent matches are attempted. The exit status is zero if no patterns are matches. Otherwise, it is the exit status of the last command executed in *list*.

- **if** *list* **then** *list* **[ elif** *list* **then** *list* **] ... [ else** *list* **] fi**

  The if *list* is executed. If its exit status is zero, the then *list* is executed. Otherwise, each elif *list* is executed in turn, and if its exit status is zero, the corresponding then *list* is executed and the command completes. Otherwise, the else *list* is executed, if present. The exit status is the exit status of the last command executed, or zero if no condition tested true.

- **while** *list* **do** *list* **done**

  **until** *list* **do** *list* **done**

The while command continuously executes the do *list* as long as the last command in *list* returns an exit status of zero. The until command is identical to the while command, except that the test is negated; the do *list* is executed as long as the last command in *list* returns a non-zero exit status. The exit status of the while and until commands is the exit status of the last do *list* command executed, or zero if none was executed.

- **[ function ]** *name* **() {** *list***; }**

  This defines a function named *name*. The body of the function is the *list* of commands between { and }. This *list* is executed whenever *name* is specified as the name of a simple command. The exit status of a function is the exit status of the last command executed in the body. (See Functions below.)

## Comments

In a non-interactive shell, a word beginning with # causes that word and all remaining characters on that line to be ignored.

## Quoting

*Quoting* is used to remove the special meaning of certain characters or words to the shell. Quoting can be used to disable special treatment for special characters, to prevent reserved words from being recognized as such, and to prevent parameter expansion.

Each of the *metacharacters* listed above under Definitions has special meaning to the shell and must be quoted if they are to represent themselves. There are three quoting mechanisms: the escape character, single quotes, and double quotes.

A non-quoted backslash (\) is the *escape character*. It preserves the literal value of the next character that follows, with the exception of <newline>. If a \<newline> pair appears, it is treated as a line continuation (that is, it is effectively ignored), if the backslash is non-quoted.

Enclosing characters in single quotes preserves the literal value of each character within the quotes. A single quote may not occur between single quotes, even when preceded by a backslash.

Enclosing characters in double quotes preserves the literal value of all characters within the quotes, with the exception of $, `, and \. The characters $ and ` retain their special meaning within double quotes. The backslash retains its special meaning only when followed by one of the following characters: $, `, ", \, or <newline>. A double quote may be quoted within double quotes by preceding it with a backslash.

The special parameters * and @ have special meaning when in double quotes (see Parameters below).

## Parameters

A *parameter* is an entity that stores values, somewhat like a variable in a conventional programming language. It can be a *name*, a number, or one of the special characters listed below under Special Parameters. For the shell's purposes, a *variable* is a parameter denoted by a *name*.

A parameter is set if it has been assigned a value. The null string is a valid value. Once a variable is set, it may be unset only by using the unset builtin command (see Shell Builtin Commands below).

A *variable* may be assigned to by a statement of the form

```
name=[value]
```

If *value* is not given, the variable is assigned the null string. All *values* undergo tilde expansion, parameter and variable expansion, command substitution, arithmetic expansion, and quote removal. If the variable has its -i attribute set (see declare below in Shell Builtin Commands) then *value* is subject to arithmetic expansion even if the $[...] syntax does not appear. Word splitting is not performed, with the exception of $@ as explained below under Special Parameters. Pathname expansion is not performed.

## Positional Parameters

A *positional parameter* is a parameter denoted by one or more digits, other than the single digit 0. Positional parameters are assigned from the shell's arguments when it is invoked, and may be reassigned using the set builtin command. The positional parameters are temporarily replaced when a shell function is executed (see Functions below).

When a positional parameter consisting of more than a single digit is expanded, it must be enclosed in braces (see Expansion below).

## Special Parameters

The shell treats several parameters specially. These parameters may only be referenced; assignment to them is not allowed.

- *****

  Expands to the positional parameters, starting from one. When the expansion occurs within double quotes, it expands to a single word with the value of each parameter separated by the first character of the IFS special variable. That is, "$*" is equivalent to "$1*c*$2*c*...", where *c* is the first character of the value of the IFS variable. If IFS is null or unset, the parameters are separated by spaces.

- **@**

  Expands to the positional parameters, starting from one. When the expansion occurs within double quotes, each parameter expands as a separate word. That is, "$@" is equivalent to "$1" "$2" ... When there are no positional parameters, "$@" and $@ expand to nothing (i.e. they are removed).

- **#**

  Expands to the number of positional parameters in decimal.

- **?**

  Expands to the status of the most recently executed foreground pipeline.

- **-**

  Expands to the current option flags as specified upon invocation, by the set builtin command, or those set by the shell itself (such as the -i flag).

- **$**

  Expands to the process ID of the shell. In a () subshell, it expands to the process ID of the current shell, not the subshell.

- **!**

  Expands to the process ID of the most recently executed background (asynchronous) command.

- **0**

Expands to the name of the shell or shell script. This is set at shell initialization. If bash is invoked with a file of commands, $0 is set to the name of that file. Otherwise, it is set to the pathname used to invoke bash, as given by argument zero.

- _

  Expands to the last argument to the previous command, after expansion. Also set to the full pathname of each command executed and placed in the environment exported to that command.

## Shell Variables

The following variables are set by the shell:

- PPID

  The process ID of the shell's parent.

- PWD

  The current working directory as set by the cd command.

- OLDPWD

  The previous working directory as set by the cd command.

- REPLY

  Set to the line of input read by the read builtin command when no arguments are supplied.

- UID

  Expands to the user ID of the current user.

- EUID

  Expands to the effective user ID of the current user.

- BASH

  Expands to the full pathname used to invoke this instance of bash.

- BASH_VERSION

  Expands to the version number of this instance of bash.

- SHLVL

  Incremented by one each time an instance of bash is started.

- RANDOM

  Each time this parameter is referenced, a random integer is generated. The sequence of random numbers may be initialized by assigning a value to RANDOM. If RANDOM is unset, it loses its special properties, even if it is subsequently reset.

- SECONDS

  Each time this parameter is referenced, the number of seconds since shell invocation is returned. If a value is assigned to SECONDS, the value returned upon subsequent references is the number of seconds since the assignment plus the value assigned. If SECONDS is unset, it loses its special properties, even if it is subsequently reset.

- LINENO

Each time this parameter is referenced, the shell substitutes a decimal number representing the current sequential line number (starting with 1) within a script or function. When not in a script or function, the value substituted is not guaranteed to be meaningful. When in a function, the value is not the number of the source line that the command appears on (that information has been lost by the time the function is executed), but is an approximation of the number of *simple commands* executed in the current function. If LINENO is unset, it loses its special properties, even if it is subsequently reset.

- OPTARG

  The value of the last option argument processed by the getopts builtin command (see Shell Builtin Commands below).

- OPTIND

  The index of the last option processed by the getopts builtin command (see Shell Builtin Commands below).

The following variables are used by the shell. In some cases, bash assigns a default value to a variable; these cases are noted below.

- IFS

  The *Internal Field Separator* that is used for word splitting after expansion and to split lines into words with the read builtin command. The default value is:

  ```
  <space><tab><newline>
  ```

- PATH

  The search path for commands. It is a colon-separated list of directories in which the shell looks for commands (see Command Execution below). The default path is system-dependent, and is set by the administrator who installs bash. A common value:

  ```
  .:/usr/gnu/bin:/usr/local/bin:/usr/ucb:/bin:/usr/bin:/etc:/usr/etc
  Note that in some circumstances, however, a leading '.' in PATH can be a security hazard.
  ```

- HOME

  The home directory of the current user; the default argument for the cd builtin command.

- CDPATH

  The search path for the cd builtin command. This is a colon-separated list of directories in which the shell looks for destination directories specified by the cd command. A sample value is:

  ```
  .:~:/usr
  ```

- ENV

  If this parameter is set when bash is executing a shell script, its value is interpreted as a filename containing commands to initialize the shell, as in .bashrc. The value of ENV is subjected to parameter expansion, command substitution, and arithmetic expansion before being interpreted as a pathname. PATH is not used to search for the resultant pathname.

- MAIL

  If this parameter is set to a filename and the MAILPATH variable is not set, bash informs the user of the arrival of mail in the specified file.

- MAILCHECK

Specifies how often (in seconds) bash checks for mail. The default is 60 seconds. When it is time to check for mail, the shell does so before prompting. If this variable is unset, the shell disables mail checking.

- MAILPATH

  A colon-separated list of pathnames to be checked for mail. The message to be printed may be specified by separating the pathname from the message with a '?'. $_ stands for the name of the current mailfile. Example:

  ```
  MAILPATH='/usr/spool/mail/bfox?"You have mail"\
  :~/shell-mail?"$_ has mail!"'
  ```

  Bash supplies a default value for this variable, but the location of the user mail files that it uses is system dependent (e.g. /usr/spool/mail/$USER).

- MAIL_WARNING

  If set, and a file that bash is checking for mail has been accessed since the last time it was checked, the message "The mail in mailfile has been read" is printed.

- PS1

  The value of this parameter is expanded (see Prompting below) and used as the primary prompt string. The default value is "bash\$ ".

- PS2

  The value of this parameter is expanded like PS1 and used as the secondary prompt string. The default is "> ".

- PS4

  The value of this parameter is expanded like PS1 and the value is printed before each command bash displays during an execution trace. The first character of PS4 is replicated multiple times, as necessary, to indicate multiple levels of indirection. The default is "+ ".

- NO_PROMPT_VARS

  If set, the decoded prompt string does not undergo further expansion (see Prompting below).

- HISTSIZE

  The number of commands to remember in the command history (see History below).

- HISTFILE

  The name of the file in which command history is saved. (See History below.)

- HISTFILESIZE

  The maximum number of lines contained in the history file. When this variable is assigned a value, the history file is truncated, if necessary, to contain no more than that number of lines.

- OPTERR

  If set to the value 1, bash displays error messages generated by the getopts builtin command (see Shell Builtin Commands below). OPTERR is initialized to 1 each time the shell is invoked or a shell script is executed.

- PROMPT_COMMAND

  If set, the value is executed as a command prior to issuing each primary prompt.

- IGNOREEOF

  ignoreeof

Controls the action of the shell on receipt of an EOF character as the sole input. If set, the value is the number of consecutive EOF characters typed before bash exits. If the variable exists but does not have a numeric value, or has no value, the default value is 10. If it does not exist, EOF signifies the end of input to the shell. This is only in effect for interactive shells.

- HOSTTYPE

  Automatically set to a string that uniquely describes the type of machine on which bash is executing. The default is system-dependent.

- TMOUT

  If set to a value greater than zero, the value is interpreted as the number of seconds to wait for input after issuing the primary prompt. Bash terminates after waiting for that number of seconds if input does not arrive.

- FCEDIT

  The default editor for the fc builtin command.

- FIGNORE

  A colon-separated list of suffixes to ignore when performing filename completion (see Readline below). A filename whose suffix matches one of the entries in FIGNORE is excluded from the list of matched filenames. A sample value is ".o:~".

- notify

  If set, bash reports terminated background jobs immediately, rather than waiting until before printing the next primary prompt.

- history_control

  If set to a value of *ignorespace*, it means don't enter lines which begin with a <space> on the history list. If set to a value of *ignoredups*, it means don't enter lines which match the last entered line. If unset, or if set to any other value than those above, all lines read by the parser are saved on the history list.

- command_oriented_history

  If set, bash attempts to save all lines of a multiple-line command in the same history entry. This allows easy re-editing of multi-line commands.

- glob_dot_filenames

  If set, bash includes filenames beginning with a '.' in the results of pathname expansion.

- allow_null_glob_expansion

  If set, bash allows pathname patterns which match no files (see Pathname Expansion below) to expand to a null string, rather than themselves.

- histchars

  The two characters which control history expansion and tokenization. The first character is the *history expansion character*, that is, the character which signals the start of a history expansion, normally '!'. The second character is the character which signifies that the remainder of the line is a comment, when found as the first character of a word.

- nolinks

  If set, the shell does not follow symbolic links when executing commands that change the current working directory. It uses the physical directory structure instead. By default, bash follows the logical chain of directories when performing commands such as cd.

- hostname_completion_file

  Contains the name of a file in the same format as /etc/hosts that should be read when the shell needs to complete a hostname. You can change the file interactively; the next time you want to complete a hostname, bash adds the contents of the new file to the already existing database.

- noclobber

  If set, bash does not overwrite an existing file with the >, >&, and <> redirection operators. This variable may be overridden when creating output files by using the redirection operator >| instead of > (see also the -C option to the set builtin command).

- auto_resume

  This variable controls how the shell interacts with the user and job control. If this variable is set, single word simple commands without redirections are treated as candidates for resumption of an existing stopped job. There is no ambiguity allowed; if there is more than one job beginning with the string typed, the job most recently accessed is selected.

- no_exit_on_failed_exec

  If this variable exists, the shell does not exit if it cannot execute the file specified in the exec command.

- cdable_vars

  If this is set, an argument to the cd builtin command that is not a directory is assumed to be the name of a variable whose value is the directory to change to.

- pushd_silent

  If set, the pushd and popd builtin commands do not print the current directory stack after successful execution.

## Expansion

Expansion is performed on the command line after it has been split into words. There are seven kinds of expansion performed: *brace expansion*, *tilde expansion*, *parameter and variable expansion*, *command substitution*, *arithmetic expansion*, *word splitting*, and *pathname expansion*.

The order of expansions is: brace expansion, tilde expansion, parameter, variable, command, and arithmetic substitution (done in a left-to-right fashion), word splitting, and pathname expansion.

Only brace expansion, word splitting, and pathname expansion can change the number of words of the expansion; other expansions expand a single word to a single word. The single exception to this is the expansion of “$@” as explained above (see Parameters).

### Brace Expansion

*Brace expansion* is a mechanism by which arbitrary strings may be generated. This mechanism is similar to *pathname expansion*, but the filenames generated need not exist. Patterns to be brace expanded take the form of an optional *preamble*, followed by a series of comma-separated strings between a pair of braces, followed by an optional *postamble*. The preamble is appended to the beginning of each string contained within the braces, and the postamble is then appended to the end of each resulting string, expanding left to right.

Brace expansions may be nested. The results of each expanded string are not sorted; left to right order is preserved. For example, a{d,c,b}e expands into ‘ade ace abe’.

Brace expansion is performed before any other expansions, and any characters special to other expansions are preserved in the result. It is strictly textual. Bash does not apply any syntactic interpretation to the context of the expansion or the text between the braces.

This construct is typically used as shorthand when the common prefix of the strings to be generated is longer than in the above example:

**mkdir /usr/local/src/bash/{old,new,dist,bugs}**

or

**chown root /usr/{ucb/{ex,edit},lib/{ex?.?*,how_ex}}**

Brace expansion introduces a slight incompatibility with traditional versions of sh, the Bourne shell. sh does not treat opening or closing braces specially when they appear as part of a word, and preserves them in the output. Bash removes braces from words as a consequence of brace expansion. For example, a word entered to sh as file{1,2} appears identically in the output. The same word is output as file1 file2 after expansion by bash. If strict compatibility with sh is desired, start bash with the -nobraceexpansion flag (see Options above) or disable brace expansion with the +o braceexpand option to the set command (see Shell Builtin Commands below).

## Tilde Expansion

If a word begins with a tilde character (‘~’), all of the characters preceding the first slash (or all characters, if there is no slash) are treated as a possible *login name*. If this *login name* is the null string, the tilde is replaced with the value of the parameter HOME. If HOME is unset, the home directory of the user executing the shell is substituted instead.

If a ‘+’ follows the tilde, the value of PWD is substituted. If a ‘-’ follows, the value of OLDPWD is used.

Each variable assignment is checked for unquoted instances of tildes following a : or =. In these cases, tilde substitution is also performed. Consequently, one may use pathnames with tildes in PATH, MAILPATH, and CDPATH, and the shell exports the expanded variables.

## Parameter Expansion

The ‘$’ character introduces parameter expansion, command substitution, or arithmetic expansion. The parameter name or symbol to be expanded may be enclosed in braces, which are optional but serve to protect the variable to be expanded from characters immediately following it which could be interpreted as part of the name.

- **${***parameter***}**

  The value of *parameter* is substituted. The braces are required when *parameter* is a positional parameter with more than one digit, or when *parameter* is followed by a character which is not to be interpreted as part of its name.

In each of the cases below, *word* is subject to tilde expansion, parameter expansion, command substitution, and arithmetic expansion. Bash tests for a parameter that is unset or null; omitting the colon results in a test only for a parameter that is unset.

- **${***parameter***:-***word***}**

  Use Default Values. If *parameter* is unset or null, the expansion of *word* is substituted. Otherwise, the value of *parameter* is substituted.

- **${***parameter***:=***word***}**

Assign Default Values. If *parameter* is unset or null, the expansion of *word* is assigned to *parameter*. The value of *parameter* is then substituted. Positional parameters and special parameters may not be assigned to in this way.

- **${***parameter***:?***word***}**

  Display error if null or unset. If *parameter* is null or unset, the expansion of *word* (or a message to that effect if *word* is not present) is written to the standard error and the shell, if it is not interactive, exits. Otherwise, the value of *parameter* is substituted.

- **${***parameter***:+***word***}**

  Use Alternate Value. If *parameter* is null or unset, nothing is substituted, otherwise the expansion of *word* is substituted.

- **${#***parameter***}**

  The length in characters of the value of *parameter* is substituted. If *parameter* is * or @, the length substituted is the length of * expanded within double quotes.

- **${***parameter***#***word***}**

  **${***parameter***##***word***}**

  The *word* is expanded to produce a pattern just as in pathname expansion. If the pattern matches the beginning of the value of *parameter*, then the expansion is the value of *parameter* with the shortest matching pattern deleted (the "#" case) or the longest matching pattern deleted (the "##" case).

- **${***parameter***%***word***}**

  **${***parameter***%%***word***}**

  The *word* is expanded to produce a pattern just as in pathname expansion. If the pattern matches a trailing portion of the value of *parameter*, then the expansion is the value of *parameter* with the shortest matching pattern deleted (the "%" case) or the longest matching pattern deleted (the "%%" case).

## Command Substitution

*Command substitution* allows the output of a command to replace the command name. There are two forms:

```
$(command)
```

or

```
‘command‘
```

Bash performs the expansion by executing *command* and replacing the command substitution with the standard output of the command, with any trailing newlines deleted.

When the old-style backquote form of substitution is used, backslash retains its literal meaning except when followed by $, **‘**, or \. When using the $(*command*) form, all characters between the parentheses make up the command; none are treated specially.

Command substitutions may be nested. To nest when using the old form, escape the inner backquotes with backslashes.

If the substitution appears within double quotes, word splitting and pathname expansion are not performed on the results.

## Arithmetic Expansion

Arithmetic expansion allows the evaluation of an arithmetic expression and the substitution of the result. The format for arithmetic expansion is:

```
$[expression]
```

The *expression* is treated as if it were within double quotes, but a double quote inside the braces is not treated specially. All tokens in the expression undergo parameter expansion, command substitution, and quote removal. Arithmetic substitutions may be nested.

The evaluation is performed according to the rules listed below under Arithmetic Evaluation. If *expression* is invalid, bash prints a message indicating failure and no substitution occurs.

## Word Splitting

The shell scans the results of parameter expansion, command substitution, and arithmetic expansion that did not occur within double quotes for *word splitting*.

The shell treats each character of IFS as a delimiter, and splits the results of the other expansions into words on these characters. If the value of IFS is exactly

```
<space><tab><newline>
```

the default, then any sequence of IFS characters serves to delimit words; otherwise each occurrence of an IFS character is treated as a delimiter. If the value of IFS is null, no word splitting occurs. IFS cannot be unset.

Explicit null arguments (”” or ”) are retained. Implicit null arguments, resulting from the expansion of *parameters* that have no values, are removed.

Note that if no expansion occurs, no splitting is performed.

## Pathname Expansion

After word splitting, bash scans each *word* for the characters *, ?, and [, unless the -f flag has been set. If one of these characters appears, then the word is regarded as a *pattern*, and replaced with an alphabetically sorted list of pathnames matching the pattern. If no matching pathnames are found, and the shell variable allow_null_glob_expansion is unset, the word is left unchanged. If the variable is set, the word is removed if no matches are found. When a pattern is used for pathname generation, the character “.” at the start of a name or immediately following a slash must be matched explicitly, unless the shell variable glob_dot_filenames is set. The slash character must always be matched explicitly. In other cases, the “.” character is not treated specially.

The special pattern characters have the following meanings:

- *

  Matches any string, including the null string.

- ?

  Matches any single character.

- [...]

  Matches any one of the enclosed characters. A pair of characters separated by a minus sign denotes a *range*; any character lexically between those two characters, inclusive, is matched. If the first character following the [ is a ! or a ^ then any character not enclosed is matched. A - or ] may be matched by including it as the first or last character in the set.

### Quote Removal

After the preceding expansions, all unquoted occurrences of the characters \, ', and " are removed.

## Redirection

Before a command is executed, its input and output may be *redirected* using a special notation interpreted by the shell. Redirection may also be used to open and close files for the current shell execution environment. The following redirection operators may appear anywhere in a *simple command* or may precede or follow a *command*. Redirections are processed in the order they appear, from left to right.

In the following descriptions, if the file descriptor number is omitted, and the first character of the redirection operator is <, the redirection refers to the standard input (file descriptor 0). If the first character of the redirection operator is >, the redirection refers to the standard output (file descriptor 1).

The word that follows the redirection operator in the following descriptions is subjected to brace expansion, tilde expansion, parameter expansion, command substitution, arithmetic expansion, quote removal, and pathname expansion. If it expands to more than one word, bash reports an error.

### Redirecting Input

Redirection of input causes the file whose name results from the expansion of *word* to be opened for reading on file descriptor *n*, or the standard input (file descriptor 0) if *n* is not specified.

The general format for redirecting input is:

**[***n***]**<*word*

### Redirecting Output

Redirection of output causes the file whose name results from the expansion of *word* to be opened for writing on file descriptor *n*, or the standard output (file descriptor 1) if *n* is not specified. If the file does not exist it is created; if it does exist it is truncated to zero size.

The general format for redirecting output is:

**[***n***]**>*word*

If the redirection operator is >|, then the variable noclobber is not consulted, and the file is created regardless of the value of noclobber (see Shell Variables above).

### Appending Redirected Output

Redirection of output in this fashion causes the file whose name results from the expansion of word to be opened for appending on file descriptor *n*, or the standard output (file descriptor 1) if *n* is not specified. If the file does not exist it is created.

The general format for appending output is:

**[***n***]**>>*word*

### Redirecting Standard Output and Standard Error

Bash allows both the standard output (file descriptor 1) and the standard error output (file descriptor 2) to be redirected to the file whose name is the expansion of word with this construct.

There are two formats for redirecting standard output and standard error:

**&**>*word*

and

**>&***word*

Of the two forms, the first is preferred. This is semantically equivalent to

*>word* **2>&1**

## Here Documents

This type of redirection instructs the shell to read input from the current source until a line containing only *word* (with no trailing blanks) is seen. All of the lines read up to that point are then used as the standard input for a command.

The format of here-documents is as follows:

**<<[-]***word*

*here-document*

*delimiter*

No parameter expansion, command substitution, pathname expansion, or arithmetic expansion is performed on *word*. If any characters in *word* are quoted, the delimiter is the result of quote removal on *word*, and the lines in the here-document are not expanded. Otherwise, all lines of the here-document are subjected to parameter expansion, command substitution, and arithmetic expansion. In the latter case, the pair \<newline> is ignored, and \ must be used to quote the characters \, $, and `.

If the redirection operator is <<-, then all leading tab characters are stripped from input lines and the line containing *delimiter*. This allows *here-documents* within shell scripts to be indented in a natural fashion.

## Duplicating File Descriptors

The redirection operator

**[***n***]<&***word*

is used to duplicate input file descriptors. If *word* expands to one or more digits, the file descriptor denoted by *n* is made to be a copy of that file descriptor. If word evaluates to -, file descriptor *n* is closed. If *n* is not specified, the standard input (file descriptor 0) is used.

The operator

**[***n***]>&***word*

is used similarly to duplicate output file descriptors. If *n* is not specified, the standard output (file descriptor 1) is used.

## Opening File Descriptors for Reading and Writing

The redirection operator

**[***n***]<>***word*

causes the file whose name is the expansion of *word* to be opened for both reading and writing on file descriptor *n*, or as the standard input and standard output if *n* is not specified.

Note that the order of redirections is significant. For example, the command

**ls** > *dirlist* **2>&1**

directs both standard output and standard error to the file *dirlist*, while the command

**ls 2>&1** > *dirlist*

directs only the standard output to file *dirlist*, because the standard error was duplicated as standard output before the standard output was redirected to *dirlist*.

## Functions

A shell function, defined as described above under Shell Grammar, stores a series of commands for later execution. However, functions are executed in the context of the current shell; no new process is created to interpret them (contrast this with the execution of a shell script). When a function is executed, the arguments to the function become the positional parameters during its execution. The special parameter # is updated to reflect the change. Positional parameter 0 is unchanged.

Variables local to the function may be declared with the local builtin command. Ordinarily, variables and their values are shared between the function and its caller.

If the builtin command return is executed in a function, the function completes and execution resumes with the next command after the function call. When a function completes, the values of the positional parameters and the special parameter # are restored to the values they had prior to function execution.

Function names may be listed with the -f option to the declare or typeset builtin commands. Functions may be exported so that subshells automatically have them defined with the -f option to the export builtin.

Functions may be recursive. No limit is imposed on the number of recursive calls.

## Aliases

The shell maintains a list of *aliases* that may be set and unset with the alias and unalias builtin commands. The first word of each command is checked to see if it has an alias. If so, that word is replaced by the text of the alias. The alias name and the replacement text may contain any valid shell input, including the *metacharacters* listed above. The first word of the replacement text is tested for aliases, but a word that is identical to an alias being expanded is not expanded a second time. This means that one may alias ls to ls -F, for instance, and bash does not try to recursively expand the replacement text. If the last character of the alias value is a *blank*, then the next command word following the alias is also checked for alias expansion.

Aliases are created and listed with the alias command, and removed with the unalias command.

There is no mechanism for using arguments in the replacement text, as in csh. If arguments are needed, a shell function should be used.

The rules concerning the definition and use of aliases are somewhat confusing. Bash always reads at least one complete line of input before executing any of the commands on that line. Aliases are expanded when a command is read, not when it is executed. Therefore, an alias definition appearing on the same line as another command does not take effect until the next line of input is read. This means that the commands following the alias definition on that line are not affected by the new alias. This behavior is also an issue when functions are executed. Aliases are expanded when the function definition is read, not when the function is executed, because a function definition is itself a compound command. As a consequence, aliases defined in a function are not available until after that function is executed. To be safe, always put alias definitions on a separate line, and do not use alias in compound commands.

Aliases are not expanded when the shell is not interactive.

Note that for almost every purpose, aliases are superseded by shell functions.

## Job Control

*Job control* refers to the ability to selectively stop (*suspend*) the execution of processes and continue (*resume*) their execution at a later point. A user typically employs this facility via an interactive interface supplied jointly by the system's terminal driver and bash.

The shell associates a *job* with each pipeline. It keeps a table of currently executing jobs, which may be listed with the jobs command. When bash starts a job asynchronously (in the *background*), it prints a line that looks like:

```
[1] 25647
```

indicating that this job is job number 1 and that the process ID of the single process in the job is 25647. Bash uses the *job* abstraction as the basis for job control.

To facilitate the implementation of the user interface to job control, the system maintains the notion of a *current terminal process group ID*. Members of this process group (processes whose process group ID is equal to the current terminal process group ID) receive keyboard-generated signals such as SIGINT. These processes are said to be in the *foreground*. *Background* processes are those whose process group ID differs from the terminal's; such processes are immune to keyboard-generated signals. Only foreground processes are allowed to read from or write to the terminal. Background processes which attempt to read from (write to) the terminal are sent a SIGTTIN (SIGTTOU) signal by the terminal driver, which, unless caught, causes the process to stop.

If the operating system on which bash is running supports job control, bash allows you to use it. Typing the *suspend* character (typically ^Z, Control-Z) while a process is running causes that process to be stopped and returns you to bash. Typing the *delayed suspend* character (typically ^Y, Control-Y) causes the process to be stopped when it attempts to read input from the terminal, and control to be returned to bash. You may then manipulate the state of this job, using the bg command to continue it in the background, the fg command to continue it in the foreground, or the kill command to kill it. A ^Z takes effect immediately, and has the additional side effect of causing pending output and typeahead to be discarded.

There are a number of ways to refer to a job in the shell. The character % introduces a job name. Job number *n* may be referred to as %*n*. A job may also be referred to using a prefix of the name used to start it, or using a substring that appears in its command line. For example, %ce refers to a stopped ce job. If a prefix matches more than one job, bash reports an error. Using %?ce, on the other hand, would refer to any job containing the string ce in its command line. If the substring matches more than one job, bash reports an error. The symbols %% and %+ refer to the shell's notion of the *current job*, which is the last job stopped while it was in the foreground. The *previous job* may be referenced using %-. In output pertaining to jobs (e.g. the output of the jobs command), the current job is always flagged with a +, and the previous job with a -.

Simply naming a job can be used to bring it into the foreground: %1 is a synonym for "fg %1", bringing job 1 from the background into the foreground. Similarly, "%1 &" resumes job 1 in the background, equivalent to "bg %1".

The shell learns immediately whenever a job changes state. Normally, bash waits until it is about to print a prompt before reporting changes in a job's status so as to not interrupt any other output. If the variable notify is set, bash reports such changes immediately. (See also the -o notify option to the set builtin command.)

If you attempt to exit bash while jobs are stopped, the shell prints a message warning you. You may then use the jobs command to inspect their status. If you do this, or try to exit again immediately, you are not warned again, and the stopped jobs are terminated.

## Signals

When bash is interactive, it ignores SIGTERM (so that **kill 0** does not kill an interactive shell), and SIGINT is caught and handled (so that **wait** is interruptible). In all cases, bash ignores SIGQUIT. If job control is in effect, bash ignores SIGTTIN, SIGTTOU, and SIGTSTP.

Synchronous jobs started by bash have signals set to the values inherited by the shell from its parent. Background jobs (jobs started with **&**) ignore SIGINT and SIGQUIT. Commands run as a result of command substitution ignore the keyboard-generated job control signals SIGTTIN, SIGTTOU, and SIGTSTP.

## Command Execution

After a command has been split into words, if it results in a simple command and an optional list of arguments, the following actions are taken.

If the command name contains no slashes, the shell attempts to locate it. If there exists a shell function by that name, that function is invoked as described above in Functions. If the name does not match a function, the shell searches for it in the list of shell builtins. If a match is found, that builtin is invoked.

If the name is neither a shell function nor a builtin, and contains no slashes, bash searches each element of the PATH for a directory containing an executable file by that name. If the search is unsuccessful, the shell prints an error message and returns a nonzero exit status.

If the search is successful, or if the command name contains one or more slashes, the shell executes the named program. Argument 0 is set to the name given, and the remaining arguments to the command are set to the arguments given, if any.

If this execution fails because the file is not in executable format, and the file is not a directory, it is assumed to be a *shell script*, a file containing shell commands. A subshell is spawned to execute it. This subshell reinitializes itself, so that the effect is as if a new shell had been invoked to handle the script, with the exception that the locations of commands remembered by the parent (see hash below under Shell Builtin Commands) are retained by the child.

If the program is a file beginning with #!, the remainder of the first line specifies an interpreter for the program. The shell executes the specified interpreter on operating systems that do not handle this executable format themselves. The arguments to the interpreter consist of a single optional argument following the interpreter name on the first line of the program, followed by the name of the program, followed by the command arguments, if any.

## Environment

When a program is invoked it is given an array of strings called the *environment*. This is a list of *name-value* pairs, of the form *name=value*.

The shell allows you to manipulate the environment in several ways. On invocation, the shell scans its own environment and creates a parameter for each name found, automatically marking it for *export* to child processes. Executed commands inherit the environment. The **export** and **declare -x** commands allow parameters and functions to be added to and deleted from the environment. If the value of a parameter in the environment is modified, the new value becomes part of the environment, replacing the old. The environment inherited by any executed command consists of the shell's initial environment, whose values may be modified in the shell, less any pairs removed by the **unset** command, plus any additions, using the **export** and **declare -x** commands.

The environment for any *simple command* or function may be augmented temporarily by prefixing it with parameter assignments, as described above in Parameters. These assignment statements affect only the environment seen by that command.

If the -k flag is set (see the **set** builtin command below), then *all* parameter assignments are placed in the environment for a command, not just those that precede the command name.

## Exit Status

For the purposes of the shell, a command which exits with a zero exit status has succeeded. An exit status of zero indicates success. A non-zero exit status indicates failure. When a command terminates on a fatal signal, bash uses the value of 128+*signal* as the exit status.

Bash itself returns the exit status of the last command executed, unless a syntax error occurs, in which case it exits with a non-zero value. See also the exit builtin command below.

## Prompting

When executing interactively, bash displays the primary prompt PS1 when it is ready to read a command, and the secondary prompt PS2 when it needs more input to complete a command. Bash allows the prompt to be customized by inserting a number of backslash-escaped special characters that are decoded as follows:

- \t — the time
- \d — the date
- \n — CRLF
- \s — the name of the shell, the basename of $0 (the portion following the final slash)
- \w — the current working directory
- \W — the basename of the current working directory
- \u — the username of the current user
- \h — the hostname
- \# — the command number of this command
- \! — the history number of this command
- \$ — if the effective UID is 0, a #, otherwise a $
- *nnn* — character code in octal
- \\ — a backslash

After the string is decoded, if the variable NO_PROMPT_VARS is not set, it is expanded via parameter expansion, command substitution, arithmetic expansion, and word splitting.

## Readline

This is the library that handles reading input when using an interactive shell, unless the **-nolineediting** option is given. By default, the line editing commands are similar to those of **emacs**. A **vi**-style line editing interface is also available.

In this section, the **emacs**-style notation is used to denote keystrokes. Control keys are denoted by C-*key*, e.g. C-*n* means Control-*N*. Similarly, meta keys are denoted by M-*key*, so M-*x* means Meta-X. (On keyboards without a meta key, M-*x* means ESC *x*, i.e. press the Escape key then the *x* key. The combination M-C-*x* means ESC-Control-*x*, or press the Escape key then hold the Control key while pressing the *x* key.)

The default key-bindings may be changed with an ~/.inputrc file. Other programs that use this library may add their own commands and bindings.

For example, placing

```
M-Control-u: universal-argument
```

or

```
C-Meta-u: universal-argument
```

into the ~/.inputrc would make M-C-u execute the command universal-argument.

The following symbolic character names are recognized:

RUBOUT, DEL, ESC, NEWLINE, SPACE, RETURN, LFD, TAB.

Placing

**set editing-mode vi**

into a ~/.inputrc file causes bash to start with a **vi**-like editing mode. The editing mode may be switched during interactive use by using the -o option to the set builtin command (see Shell Builtin Commands below).

You can have readline use a single line for display, scrolling the input between the two borders by placing

**set horizontal-scroll-mode On**

into a ~/.inputrc file.

The following is a list of the names of the commands and the default key-strokes to get them.

## Commands for Moving

- beginning-of-line (C-a)

  Move to the start of the current line.

- end-of-line (C-e)

  Move to the end of the line.

- forward-char (C-f)

  Move forward a character.

- backward-char (C-b)

  Move back a character.

- forward-word (M-f)

  Move forward to the end of the next word.

- backward-word (M-b)

  Move back to the start of this, or the previous, word.

- clear-screen (C-l)

Clear the screen leaving the current line at the top of the screen.

### Commands for Manipulating the History

- accept-line (Newline, Return)

  Accept the line regardless of where the cursor is. If this line is non-empty, add it to the history list according to the state of the history_control variable. If this line was a history line, then restore the history line to its original state.

- previous-history (C-p)

  Fetch the previous command from the history list, moving back in the list.

- next-history (C-n)

  Fetch the next command from the history list, moving forward in the list.

- beginning-of-history (M-<)

  Move to the first line in the history, the first line entered.

- end-of-history (M->)

  Move to the end of the input history, i.e., the line you are entering.

- reverse-search-history (C-r)

  Search backward starting at the current line and moving ‘up’ through the history as necessary. This is an incremental search.

- forward-search-history (C-s)

  Search forward starting at the current line and moving ‘down’ through the history as necessary.

- shell-expand-line (M-C-e)

  Expand the line the way the shell does when it reads it. This performs alias and history expansion. See History below.

- insert-last-argument (M-., M-_)

  Insert the last argument to the previous command (the last word on the previous line).

- operate-and-get-next (C-O)

  Accept the current line for execution and fetch the next line relative to the current line from the history file for editing.

### Commands for Changing Text

- delete-char (C-d)

  Delete the character under the cursor. If the cursor is at the beginning of the line, and there are no characters in the line, and the last character typed was not C-d, then return EOF.

- backward-delete-char (Rubout)

  Delete the character behind the cursor. A numeric arg says to kill the characters instead of deleting them.

- quoted-insert (C-q, C-v)

  Add the next character that you type to the line verbatim. This is how to insert characters like C-q, for example.

- tab-insert (M-TAB)

  Insert a tab character.

- self-insert (a, b, A, 1, !, ...)

  Insert the character typed.

- transpose-chars (C-t)

  Drag the character before point forward over the character at point. Point moves forward as well. If point is at the end of the line, then transpose the two characters before point. Negative arguments don't work.

- transpose-words (M-t)

  Drag the word behind the cursor past the word in front of the cursor moving the cursor over that word as well.

- upcase-word (M-u)

  Uppercase the current (or following) word. With a negative argument, do the previous word, but do not move point.

- downcase-word (M-l)

  Lowercase the current (or following) word. With a negative argument, do the previous word, but do not move point.

- capitalize-word (M-c)

  Capitalize the current (or following) word. With a negative argument, do the previous word, but do not move point.

### Killing and Yanking

- kill-line (C-k)

  Kill the text from the current cursor position to the end of the line. This saves the killed text on the kill-ring (see below).

- backward-kill-line

  Kill backward to the beginning of the line. This is normally unbound, in favor of unix-line-discard, which emulates the behavior of the standard Unix terminal driver.

- kill-word (M-d)

  Kill from the cursor to the end of the current word, or if between words, to the end of the next word.

- backward-kill-word (M-Rubout)

  Kill the word behind the cursor.

- unix-line-discard (C-u)

  Do what C-u used to do in Unix line input. We save the killed text on the kill-ring, though.

- unix-word-rubout (C-w)

  Do what C-w used to do in Unix line input. The killed text is saved on the kill-ring. This is different than backward-kill-word because the word boundaries differ.

- yank (C-y)

Yank the top of the kill ring into the buffer at point.

- yank-pop (M-y)

  Rotate the kill-ring, and yank the new top. Only works following yank or yank-pop.

### Arguments

- digit-argument (M-0, M-1, ..., M--)

  Add this digit to the argument already accumulating, or start a new argument. M-- starts a negative argument.

- universal-argument

  Do what C-u does in emacs. By default, this is not bound to a key.

### Completing

- complete (TAB)

  Attempt to perform completion on the text before point. Bash attempts completion treating the text as a variable (if the text begins with $), username (if the text begins with ~), hostname (if the text begins with @), or command (including aliases and functions) in turn. If none of these produces a match, filename completion is attempted.

- possible-completions (M-?)

  List the possible completions of the text before point.

- complete-filename (M-/)

  Attempt filename completion on the text before point.

- possible-filename-completions (C-x /)

  List the possible completions of the text before point, treating it as a filename.

- complete-username (M-~)

  Attempt completion on the text before point, treating it as a username.

- possible-username-completions (C-x ~)

  List the possible completions of the text before point, treating it as a username.

- complete-variable (M-$)

  Attempt completion on the text before point, treating it as a shell variable.

- possible-variable-completions (C-x $)

  List the possible completions of the text before point, treating it as a shell variable.

- complete-hostname (M-@)

  Attempt completion on the text before point, treating it as a hostname.

- possible-hostname-completions (C-x @)

  List the possible completions of the text before point, treating it as a hostname.

### Miscellaneous

- abort (C-g)

Abort the current editing command and ring the terminal's bell.

- do-uppercase-version (M-a, M-b, ...)

  Run the command that is bound to the uppercased key.

- prefix-meta (ESC)

  Metafy the next character typed. This is for people without a meta key. ESC f is equivalent to Meta-f.

- undo (C-_)

  Incremental undo, separately remembered for each line.

- revert-line (M-r)

  Undo all changes made to this line. This is like typing the undo command enough times to get back to the beginning.

- display-shell-version (C-x C-v)

  Display version information about the current instance of bash.

- emacs-editing-mode (C-e)

  When in vi editing mode, this causes a switch to emacs editing mode.

- vi-editing-mode (M-C-j or M-C-m)

  When in emacs editing mode, this causes a switch to vi editing mode.

## History

The shell supports a history expansion feature that is similar to the history expansion in csh. This section describes what syntax features are available.

History expansion is performed immediately after a complete line is read, before the shell breaks it into words. It takes place in two parts. The first is determining which line from the previous history to use during substitution. The second is to select portions of that line for inclusion into the current one. The line selected from the previous history is the *event*, and the portions of that line that are acted upon are *words*. The line is broken into words in the same fashion as when reading input, so that several English, or Unix, words surrounded by quotes are considered as one word. Only backslash (\) can quote the history escape character, which is ! by default.

### Event Designators

An event designator is a reference to a command line entry in the history list.

- !

  Start a history substitution, except when followed by a <space>, <tab>, <newline>, =, or (.

- !!

  Refer to the previous command. This is a synonym for '!-1'.

- !*n*

  Refer to command line *n*.

- !-*n*

  Refer to the current command line minus *n*.

- !*string*

  Refer to the most recent command starting with *string*.

- !?*string*[?]

  Refer to the most recent command containing *string*.

## Word Designators

A : separates the event specification from the word designator. It can be omitted if the word designator begins with a ^, $, *, or %. Words are numbered from the beginning of the line, with the first word being denoted by a 0 (zero).

- #

  The entire command line typed so far. This means the current command, not the previous command, so it really isn't a word designator, and doesn't belong in this section.

- 0 (zero)

  The zeroth word. For the shell, this is the command word.

- *n*

  The *n*th word.

- ^

  The first argument. That is, word 1.

- $

  The last argument.

- %

  The word matched by the most recent '?string?' search.

- *x*-*y*

  A range of words; '-*y*' abbreviates '0-*y*'.

- *

  All of the words but the zeroth. This is a synonym for '1-$'. It is not an error to use * if there is just one word in the event; the empty string is returned in that case.

## Modifiers

After the optional word designator, you can add a sequence of one or more of the following modifiers, each preceded by a ':'.

- h

  Remove a trailing pathname component, leaving only the head.

- r

  Remove a trailing suffix of the form ".*xxx*", leaving the basename.

- e

  Remove all but the suffix.

- t

Remove all leading pathname components, leaving the tail.

- p

  Print the new command but do not execute it. This takes effect immediately, so it should be the last specifier on the line.

## Arithmetic Evaluation

The shell allows arithmetic expressions to be evaluated, under certain circumstances (see the let builtin command and Arithmetic Expansion). Evaluation is done in long integers with no check for overflow, though division by 0 is trapped and flagged as an error. The following list of operators is grouped into levels of equal-precedence operators. The levels are listed in order of decreasing precedence.

- -

  unary minus

- !

  logical NOT

- * / %

  multiplication, division, remainder

- + -

  addition, subtraction

- <= >= < >

  comparison

- == !=

  equality and inequality

- =

  assignment

Shell variables are allowed as operands; parameter expansion is performed before the expression is evaluated. The value of a parameter is coerced to a long integer within an expression. A shell variable need not have its integer attribute turned on to be used in an expression.

Operators are evaluated in order of precedence. Subexpressions in parentheses are evaluated first and may override the precedence rules above.

## Shell Builtin Commands

- **: [***arguments***]**

  No effect; the command does nothing beyond expanding *arguments* and performing any specified redirections. A zero exit code is returned.

- **.** *filename*

  **source** *filename*

  Read and execute commands from *filename* in the current shell environment and return the exit status of the last command executed from *filename*. Pathnames in PATH are used to find the directory containing *filename*, if *filename* does not contain a slash. The file searched for in PATH

need not be executable. The current directory is searched if no file is found in PATH. The return status is the status of the last command exited within the script (true if no commands are executed), and false if *filename* is not found.

- **alias [***name***[=***value***] ...]**

  alias with no arguments prints the list of aliases in the form *name=value* on standard output. When arguments are supplied, an alias is defined for each *name* whose *value* is given. A trailing space in *value* causes the next word to be checked for alias substitution when the alias is expanded. alias returns true unless a *name* is given for which no alias has been defined.

- **bg [***jobspec***]**

  Place *jobspec* in the background, as if it had been started with &. If *jobspec* is not present, the shell's notion of the *current job* is used.

- **bind [-lvd] [-q** *name***]**

  **bind -f** *filename*

  **bind** *keyseq***:***function-name*

  Display current readline key and function bindings, or bind a key sequence to a readline function or macro. The binding syntax accepted is identical to that of .inputrc, but each binding must be passed as a separate argument; e.g.

  **'"\C-x\C-r": re-read-init-file'**

  Options, if supplied, have the following meanings:

  — **-**

    lList the names of all readline functions

  — **-v**

    List current function names and bindings

  — **-d**

    Dump function names and bindings in such a way that they can be re-read

  — **-f** *filename*

    Read key bindings from *filename*

  — **-q** *function*

    Query about which keys invoke the named *function*

- **break [***n***]**

  Exit from within a for, while, or until loop. If *n* is specified, break *n* levels. *n* must be >= 1. If *n* is greater than the number of enclosing loops, all enclosing loops are exited. The return value is 0 unless the shell is not executing a loop when break is executed.

- **builtin [***shell-builtin* **[***arguments***]]**

  Execute the specified shell builtin command, passing it *arguments*, and return its exit status. This is useful when you wish to define a function whose name is the same as a shell builtin, but need the functionality of the builtin within the function itself. The cd builtin is commonly redefined this way.

- **cd [***dir***]**

Change the current directory to *dir*. The variable HOME is the default *dir*. The variable CDPATH defines the search path for the directory containing *dir*. Alternative directory names are separated by a colon (:). A null directory name in CDPATH is the same as the current directory, i.e. ".". If *dir* begins with a slash (/), then CDPATH is not used. An argument of - is equivalent to $OLDPWD. The return value is true if the directory was successfully changed; false otherwise.

- **command [-p] [***command* **[***arg* **...]]**

  Run *command* with *args* suppressing the normal shell function lookup. Only builtin commands or commands found in the PATH are executed. If the -p option is given, the search for *command* is performed using a default value for PATH that is guaranteed to find all of the standard utilities. An argument of -- disables option checking for the rest of the arguments. If an error occurred or *command* cannot be found, the exit status is 127. Otherwise, the exit status of the command builtin is the exit status of *command*.

- **continue [***n***]**

  Resume the next iteration of the enclosing for, while, or until loop. If *n* is specified, resume at the *n*th enclosing loop. *n* must be >= 1. If *n* is greater than the number of enclosing loops, the last enclosing loop (the 'top-level' loop) is resumed. The return value is 0 unless the shell is not executing a loop when continue is executed.

- **declare [-frxi] [***name***[=***value***]]**

  **typeset [-frxi] [***name***[=***value***]]**

  Declare variables and/or give them attributes. If no *names* are given, then display the values of variables instead.

  — **-f**

  Use function names only

  — **-r**

  Make *names* readonly. These names cannot then be assigned values by subsequent assignment statements.

  — **-x**

  Mark *names* for export to subsequent commands via the environment.

  — **-i**

  The variable is treated as an integer; arithmetic evaluation (see Arithmetic Evaluation ) is performed when the variable is assigned a value.

  Using '+' instead of '-' turns off the attribute instead. When used in a function, makes *names* local, as with the local command.

- **dirs [-l]**

  Display the list of currently remembered directories. Directories are added to the list with the pushd command; the popd command moves back up through the list. The -l option produces a longer listing; the default listing format uses a tilde to denote the home directory.

- **echo [-ne] [***arg* **...]**

  Output the *args*, separated by spaces. If -n is specified, the trailing newline is suppressed. If the -e option is given, interpretation of the following backslash-escaped characters is enabled:

  \a — alert (bell)

  \b — backspace

\c — suppress trailing newline

\f — form feed

\n — new line

\r — carriage return

\t — horizontal tab

\v — vertical tab

\\ — backslash

*nnn* — the character whose ASCII code is *nnn* (octal)

- **enable [-n] [***name* **...]**

  Enable and disable builtin shell commands. This allows the execution of a disk command which has the same name as a shell builtin without specifying a full pathname. If -n is used, each *name* is disabled; otherwise, *names* are enabled. For example, to use the test found in PATH instead of the shell builtin version, type enable -n test.

- **eval [***arg* **...]**

  The *args* are read and concatenated together into a single command. This command is then read and executed by the shell, and its exit status is returned as the value of the eval command. If there are no *args*, or only null arguments, eval returns true.

- **exec [[-]** *command* **[***arguments***]]**

  If *command* is specified, it replaces the shell. No new process is created. The *arguments* become the arguments to *command*. If the first argument is -, the shell places a dash in the zeroth arg passed to *command*. This is what login does. If the file cannot be executed for some reason, the shell exits, unless the shell variable no_exit_on_failed_exec exists. If *command* is not specified, any redirections take effect in the current shell.

- **exit [***n***]**

  **bye [***n***]**

  Cause the shell to exit with a status of *n*. If *n* is omitted, the exit status is that of the last command executed. A trap on EXIT is executed before the shell terminates.

- **export [-npf] [***name***[=***word***]] ...**

  The supplied *names* are marked for automatic export to the environment of subsequently executed commands. If the -f option is given, the *names* refer to functions. If no *names* are given, or if the -p option is supplied, a list of all names that are exported in this shell is printed. The -n option causes the export property to be removed from the named variables. An argument of -- disables option checking for the rest of the arguments. export returns an exit status of true unless an illegal option is encountered.

- **fc [-e** *ename***] [-nlr] [***first***] [***last***]**

  **fc -s [***pat=rep***] [***cmd***]**

  Fix Command. In the first form, a range of commands from *first* to *last* is selected from the history list. *First* and *last* may be specified as a string (to locate the last command beginning with that string) or as a number (an index into the history list, where a negative number is used as an offset from the current command number). If *last* is not specified it is set to the current command for listing (so that fc -l -10 prints the last 10 commands) and to *first* otherwise. If *first* is not specified it is set to the previous command for editing and -16 for listing.

The -n flag suppresses the command numbers when listing. The -r flag reverses the order of the commands. If the -l flag is given, the commands are listed on standard output. Otherwise, the editor given by *ename* is invoked on a file containing those commands. If *ename* is not given, the value of the FCEDIT variable is used, and the value of EDITOR if FCEDIT is not set. If neither variable is set, vi is used. When editing is complete, the edited commands are echoed and executed.

In the second form, the command is re-executed after the substitution *old=new* is performed. A useful alias to use with this is "r=fc -s", so that typing "r cc" runs the last command beginning with "cc" and typing "r" re-executes the last command.

- **fg [***jobspec***]**

  Place *jobspec* in the foreground, and make it the current job. If *jobspec* is not present, the shell's notion of the *current job* is used.

- **getopts** *optstring name* **[***args***]**

  getopts is used by shell procedures to parse positional parameters. *optstring* contains the option letters to be recognized; if a letter is followed by a colon, the option is expected to have an argument, which should be separated from it by white space. Each time it is invoked, getopts places the next option in the shell variable *name*, initializing *name* if it does not exist, and the index of the next argument to be processed into the variable OPTIND. OPTIND is initialized to 1 each time the shell or a shell script is invoked. When an option requires an argument, getopts places that argument into the variable OPTARG. The shell does not reset OPTIND automatically; it must be manually reset between multiple calls to getopts within the same shell invocation if a new set of parameters is to be used.

  getopts can report errors in two ways. If the first character of *optstring* is a colon, *silent* error reporting is used. In normal operation diagnostic messages are printed when illegal options or missing option arguments are encountered. If the variable OPTERR is set to 0, no error message is displayed, even if the first character of *optstring* is not a colon.

  If an illegal option is seen, getopts places ? into *name* and, if not silent, prints an error message and unsets OPTARG. If getopts is silent, the option character found is placed in OPTARG and no diagnostic message is printed.

  If a required argument is not found, and getopts is not silent, a question mark (?) is placed in *name*, OPTARG is unset, and a diagnostic message is printed. If getopts is silent, then a colon (:) is placed in *name* and OPTARG is set to the option character found.

  getopts normally parses the positional parameters, but if more arguments are given in *args*, getopts parses those instead. getopts returns true if an option, specified or unspecified, is found. It returns false if the end of options is encountered or an error occurs.

- **hash [-r] [***name***]**

  For each *name*, the full pathname of the command is determined and remembered. The -r option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is printed. An argument of -- disables option checking for the rest of the arguments. The return status is true unless a *name* is not found or an illegal option is supplied.

- **help [***pattern***]**

  Display helpful information about builtin commands. If *pattern* is specified, help gives detailed help on all commands matching *pattern*; otherwise a list of the builtins is printed.

- **history [***n***]**

  **history -rwan [***filename***]**

With no options, display the command history list with line numbers. Lines listed with with a * have been modified. An argument of *n* lists only the last *n* lines. If a non-option argument is supplied, it is used as the name of the history file; if not, the value of HISTFILE (default ~/.bash_history) is used. Options, if supplied, have the following meanings:

-a — Append the "new" history lines (history lines entered since the beginning of the current bash session) to the history file

-n — Read the history lines not already read from the history file into the current history list. These are lines appended to the history file since the beginning of the current bash session.

-r — read the contents of the history file and use them as the current history

-w — write the current history to the history file, overwriting the history file's contents.

- **jobs [-lnp] [** *jobspec* **... ]**

  **jobs -x** *command* **[** *args* **... ]**

  The first form lists the active jobs. The -l option lists process IDs in addition to the normal information; the -p option lists only the process ID of the job's process group leader. The -n option displays only jobs that have changed status since last notified. If *jobspec* is given, output is restricted to information about that job.

  If the -x option is supplied, jobs replaces any *jobspec* found in *command* or *args* with the corresponding process group ID, and executes *command* passing it *args*.

- **kill [-s** *sigspec* **|** *-sigspec***] [***pid* **|** *jobspec***] ...**

  **kill -l [***signum***]**

  Send the signal named by *sigspec* to the processes named by *pid* or *jobspec*. *sigspec* is either a signal name such as SIGKILL or a signal number. If *sigspec* is a signal name, the name is case insensitive and may be given with or without the SIG prefix. If *sigspec* is not present, then SIGTERM is assumed. An argument of -l lists the signal names. If any arguments are supplied when -l is given, the names of the specified signals are listed. An argument of -- disables option checking for the rest of the arguments. kill returns true if at least one signal was successfully sent, or false if an error occurs.

- **let** *arg* **[***arg* **...]**

  Each *arg* is an arithmetic expression to be evaluated (see Arithmetic Evaluation). If the last *arg* evaluates to 0, let returns 1; 0 is returned otherwise.

- **local [***name***[=***value***]]**

  Create a local variable named *name*, and assign it *value*. When local is used within a function, it causes the variable *name* to have a visible scope restricted to that function and its children. With no operands, local writes a list of local variables to the standard output. It is an error to use local when not within a function.

- **logout**

  Exit a login shell.

- **popd [+/-***n***]**

  Removes entries from the directory stack. With no arguments, removes the top directory from the stack, and performs a cd to the new top directory.

  +*n* — removes the *n*th entry counting from the left of the list shown by dirs, starting with zero. For example: popd +0 removes the first directory, popd +1 the second.

-*n* — removes the *n*th entry counting from the right of the list shown by dirs, starting with zero. For example: popd -0 removes the last directory, popd -1 the next to last.

If the variable pushd_silent is unset and the popd command is successful, a dirs is performed as well.

- **pushd** *dir*

  **pushd** +/-*n*

  Adds a directory to the top of the directory stack, or rotates the stack, making the new top of the stack the current working directory. With no arguments, exchanges the top two directories.

  +*n* — Rotates the stack so that the *n*th directory (counting from the left of the list shown by dirs) is at the top.

  -*n* — Rotates the stack so that the *n*th directory (counting from the right) is at the top.

  *dir* — adds *dir* to the directory stack at the top, making it the new current working directory.

  If the variable pushd_silent is not set and the pushd command is successful, a dirs is performed as well.

- **pwd**

  Print the absolute pathname of the current working directory. The path printed contains no symbolic links (but see the description of nolinks under Shell Variables above).

- **read [-r] [***name* **...]**

  One line is read from the standard input, and the first word is assigned to the first name, the second word to the second name, and so on, with leftover words assigned to the last name. Only the characters in IFS are recognized as word delimiters. The return code is zero, unless end-of-file is encountered. If the -r option is given, a backslash-newline pair is not ignored, and the backslash is considered to be part of the line.

- **readonly [-pf] [***name* **...]**

  The given names are marked readonly and the values of these names may not be changed by subsequent assignment. If the -f option is supplied, the functions corresponding to the names are so marked. If no arguments are given, or if the -p option is supplied, a list of all readonly names is printed. An argument of -- disables option checking for the rest of the arguments.

- **return [***n***]**

  Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed in the function body. If used outside a function, but during execution of a script by the . (source) command, it causes the shell to stop executing that script and return either *n* or the exit status of the last command executed within the script as the exit status of the script.

- **set [-aefhknotuvxldCH] [***arg* **...]**

  -a — Automatically mark variables which are modified or created for export to the environment of subsequent commands.

  -e — Exit immediately if a *simple-command* (see Shell Grammar above) exits with a non-zero status. The shell does not exit if the command that fails is part of an until or while loop, part of an if statement, part of a && or || list, or if the command's return value is being inverted by means of !.

  -f — Disable pathname expansion.

  -h — Locate and remember function commands as functions are defined. Function commands are normally looked up when the function is executed.

-k — All keyword arguments are placed in the environment for a command, not just those that precede the command name.

-m — Monitor mode. Job control is enabled. This flag is on by default for interactive shells on systems that support it (see Job Control above). Background processes run in a separate process group and a line containing their exit status is printed upon their completion.

-n — Read commands but do not execute them. This may be used to check a shell script for syntax errors. This is ignored for interactive shells.

-o — option-name. The option-name can be one of the following:

— allexport

Same as -a.

— braceexpand

The shell performs curly brace expansion (see Brace Expansion above). This is on by default.

— emacs

Use an emacs-style command line editing interface.

— errexit

Same as -e.

— histexpand

Same as -H.

— ignoreeof

The effect is as if the shell command IGNOREEOF=10 had been executed (see Shell Variables above).

— monitor

Same as -m.

— noclobber

Same as -C.

— noexec

Same as -n.

— noglob

Same as -f.

— nohash

Same as -d.

— notify

The effect is as if the shell command notify= had been executed (see Shell Variables above).

— nounset

Same as -u.

— verbose

Same as -v.

— vi

Use a vi-style command line editing interface.

— xtrace

Same as -x.

If no *option-name* is supplied, the values of the current options are printed.

-t — Exit after reading and executing one command.

-u — Treat unset variables as an error when performing parameter expansion. If expansion is attempted on an unset variable, the shell prints an error message, and, if not interactive, exits with a non-zero status.

-v — Print shell input lines as they are read.

-x — After expanding each simple-command, bash displays the expanded value of PS4, followed by the command and its expanded arguments.

-l — Save and restore the binding of name in a for name [in word] command (see Shell Grammar above).

-d — Disable the hashing of commands that are looked up for execution. Normally, commands are remembered in a hash table, and once found, do not have to be looked up again.

-C — The effect is as if the shell command noclobber= had been executed (see Shell Variables above).

-H — Enable ! style history substitution. This flag is on by by default.

-- — If no arguments follow this flag, then the positional parameters are unset. Otherwise, the positional parameters are set to the args, even if some of them begin with a -.

- — Signal the end of options, cause all remaining args to be assigned to the positional parameters. The -x and -v options are turned off. If there are no args, the positional parameters remain unchanged.

Using + rather than - causes these flags to be turned off. The flags can also be specified as options to an invocation of the shell. The current set of flags may be found in $-. After the option arguments are processed, the remaining args are treated as values for the positional parameters and are assigned, in order, to $1, $2, ... $9. If no options or args are supplied, all shell variables are printed. The return status is always true unless an illegal option is encountered.

- **shift [*n*]**

  The positional parameters from *n*+1 ... are renamed to $1 .... If *n* is not given, it is assumed to be 1. The exit status is 1 if *n* is greater than $#; otherwise 0.

- **suspend [-f]**

  Suspend the execution of this shell until it receives a SIGCONT signal. The -f option says not to complain if this is a login shell; just suspend anyway.

- **test** *expr*

  **[** *expr* **]**

  Return a status of 0 (true) or 1 (false) depending on the evaluation of the conditional expression *expr*. Expressions may be unary or binary. Unary expressions are often used to examine the status of a file. There are string operators and numeric comparison operators as well.

  — -b *file*

  True if *file* exists and is block special.

— -c *file*

True if *file* exists and is character special.

— -d *file*

True if *file* exists and is a directory.

— -e *file*

True if *file* exists

— -f *file*

True if *file* exists and is a regular file.

— -g *file*

True if *file* exists and is set-group-id.

— -k *file*

True if *file* has its “sticky” bit set.

— -L *file*

True if *file* exists and is a symbolic link.

— -p *file*

True if *file* exists and is a named pipe.

— -r *file*

True if *file* exists and is readable.

— -s *file*

True if *file* exists and has a size greater than zero.

— -S *file*

True if *file* exists and is a socket.

— -t [*fd*]

True if *fd* is opened on a terminal. If *fd* is omitted, it defaults to 1 (standard output).

— -u *file*

True if *file* exists and its set-user-id bit is set.

— -w *file*

True if *file* exists and is writeable.

— -x *file*

True if *file* exists and is executable.

— -O *file*

True if *file* exists and is owned by the effective user ID.

— -G *file*

True if *file* exists and is owned by the effective group ID.

— *file1* -nt *file2*

True if *file1* is newer (according to modification date) than *file2*.

— *file1* -ot *file2*

True if *file1* is older than *file2*.

— *file1* -ef *file*

True if *file1* and *file2* have the same device and inode numbers.

— -z *string*

True if the length of *string* is zero.

— -n *string*

*string*

True if the length of *string* is non-zero.

— *string1 = string2*

True if the *strings* are equal.

— *string1* != *string2*

True if the *strings* are not equal.

— ! *expr*

True if *expr* is false.

— *expr1* -a *expr2*

True if both *expr1* AND *expr2* are true.

— *expr1* -o *expr2*

True if either *expr1* OR *expr2* is true.

— *arg1 OP arg2*

*OP* is one of -eq, -ne, -lt, -le, -gt, or -ge. These arithmetic binary operators return true if *arg1* is equal, not-equal, less-than, less-than-or-equal, greater-than, or greater-than-or-equal than *arg2*, respectively. *Arg1* and *arg2* may be positive integers, negative integers, or the special expression -l *string*, which evaluates to the length of *string*.

- **times**

Print the accumulated user and system times for the shell and for processes run from the shell.

- **trap [***arg***] [***sigspec***]**

The command *arg* is to be read and executed when the shell receives signal(s) *sigspec*. If *arg* is absent or -, all specified signals are are reset to their original values (the values they had upon entrance to the shell). If arg is the null string this signal is ignored by the shell and by the commands it invokes. *sigspec* is either a signal name in <signal.h>, or a signal number. If *sigspec* is EXIT (0) the command *arg* is executed on exit from the shell. With no arguments, trap prints the list of commands associated with each signal number. The -l option causes the shell to print a list of signal names and their corresponding numbers. An argument of -- disables option checking for the rest of the arguments. Signals ignored upon entry to the shell cannot be trapped or reset. Trapped signals are reset to their original values in a child process when it is created. The return status is false if either then trap name or number is invalid; otherwise trap returns true.

- **type [-all] [-type | -path] [***name* **...]**

With no options, indicate how each *name* would be interpreted if used as a command name. If the -type flag is used, type prints a phrase which is one of *alias, keyword, function, builtin,* or *file* if *name* is an alias, shell reserved word, function, builtin, or disk file, respectively. If the name is

not found, then nothing is printed, and an exit status of false is returned. If the -path flag is used, type either returns the name of the disk file that would be executed if *name* were specified as a command name, or nothing if -type would not return *file*. If a command is hashed, -path prints the hashed value, not necessarily the file that appears first in PATH. If the -all flag is used, type prints all of the places that contain an executable named *name*. This includes aliases and functions, if and only if the -path flag is not also used. The table of hashed commands is not consulted when using -all. type accepts -a, -t, and -p in place of -all, -type, and -path, respectively. An argument of -- disables option checking for the rest of the arguments. type returns true if any of the arguments are found, false if none are found.

- **ulimit [-SHacdfmstpn [***limit***]]**

  ulimit provides control over the resources available to the shell and to processes started by it, on systems that allow such control. The value of *limit* can be a number in the unit specified for the resource, or the value unlimited. The H and S options specify that the hard or soft limit is set for the given resource. A hard limit cannot be increased once it is set; a soft limit may be increased up to the value of the hard limit. If neither H nor S is specified, the command applies to the soft limit. If *limit* is omitted, the current value of the soft limit of the resource is printed, unless the H option is given. When more than one resource is specified, the limit name and unit is printed before the value. Other options are interpreted as follows:

  -a — all current limits are reported

  -c — the maximum size of core files created

  -d — the maximum size of a process's data segment

  -f — the maximum size of files created by the shell

  -m — the maximum resident set size

  -s — the maximum stack size

  -t — the maximum amount of cpu time in seconds

  -p — the pipe size in 512-byte blocks (this may not be set)

  -n — the maximum number of open file descriptors (most systems do not allow this value to be set, only displayed)

  An argument of -- disables option checking for the rest of the arguments. If *limit* is given, it is the new value of the specified resource (the -a option is display only). If no option is given, then -f is assumed. Values are in 1024-byte increments, except for -t, which is in seconds, and -p, which is in units of 512-byte blocks.

- **umask [-S] [***mode***]**

  The user file-creation mask is set to *mode*. If *mode* begins with a digit, it is interpreted as an octal number; otherwise it is interpreted as a symbolic mode mask similar to that accepted by chmod(1). If *mode* is omitted, or if the -S option is supplied, the current value of the mask is printed. The -S option causes the mask to be printed in symbolic form; the default output is an octal number. An argument of -- disables option checking for the rest of the arguments.

- **unalias [***name* **...]**

  Remove *name*s from the list of defined aliases. The return value is true unless *name* is not a defined alias.

- **unset [-fv] [***name* **...]**

For each *name*, remove the corresponding variable or, given the -f option, function. An argument of -- disables option checking for the rest of the arguments. Note that PATH, IFS, PPID, PS1, PS2, UID, and EUID cannot be unset. If any of RANDOM, SECONDS, or LINENO are unset, they lose their special properties, even if they are subsequently reset. The exit status is true unless the variable *name* does not exist or is non-unsettable.

- **wait [*n*]**

  Wait for the specified process and report its termination status. *n* may be a process ID or a job specification; if a job spec is given, all processes in that job's pipeline are waited for. If *n* is not given, all currently active child processes are waited for, and the return code is zero.

## Invocation

A *login shell* is one whose first character of argument zero is a -, or one started with the -login flag.

An *interactive shell* is one whose standard input and output are both connected to terminals (as determined by isatty(3)), or one started with the -i flag. PS1 is set and $- includes i if bash is interactive, allowing a way to test this state from a shell script or a startup file.

Login shells:

- On login:

  if /etc/profile exists, source it.

  if ~/.bash_profile exists, source it,

  else if ~/.bash_login exists, source it,

  else if ~/.profile exists, source it.

- On logout:

  if ~/.bash_logout exists, source it.

Non-login interactive shells:

- On startup:

  if ~/.bashrc exists, source it.

Non-interactive shells:

- On startup:

  if the environment variable ENV is non-null, expand it and source the file it names.

# See Also

*The Gnu Readline Library*, Brian Fox

*The Gnu History Library*, Brian Fox

*A System V Compatible Implementation of 4.2BSD Job Control*, David Lennert

*How to wear weird pants for fun and profit*, Brian Fox

sh(1), ksh(1), csh(1)

# Files

/bin/bash — The bash executable

/etc/profile — The system-wide initialization file, executed for login shells

~/.bash_profile — The personal initialization file, executed for login shells

~/.bashrc — The individual per-interactive-shell startup file

~/.inputrc — Individual Readline initialization file

# Authors

Brian Fox, Free Software Foundation (primary author)

bfox@ai.MIT.Edu

Chet Ramey, Case Western Reserve University

chet@ins.CWRU.Edu

# Bug Reports

If you find a bug in bash, you should report it. But first, you should make sure that it really is a bug, and that it appears in the latest version of bash that you have.

Once you have determined that a bug actually exists, mail a bug report to bash-maintainers@ai.MIT.Edu. If you have a fix, you are welcome to mail that as well! Suggestions and ‘philosophical’ bug reports may be mailed to bug-bash@ai.MIT.Edu or posted to the Usenet newsgroup gnu.bash.bug.

ALL bug reports should include:

- The version number of bash
- The hardware and operating system
- The compiler used to compile
- A description of the bug behavior
- A short script or ‘recipe’ which exercises the bug

Comments and bug reports concerning this manual page should be directed to chet@ins.CWRU.Edu.

# Bugs

It’s too big and too slow.

There are some subtle differences between bash and traditional versions of sh, mostly because of the POSIX specification.

Aliases are confusing in some uses.