April 1993
Order Number: 312486-001

PARAGON™ OSF/1 COMMANDS REFERENCE MANUAL

Intel® Corporation

REV.	REVISION HISTORY	DATE
-001	Original Issue	4/93

LIMITED RIGHTS

Preface

This manual describes the Paragon™ OSF/1 system commands. The system commands let you run applications and manage partitions. You issue system commands at your shell prompt.

The manual assumes you know how to use the OSF/1 operating system.

Organization

This manual contains a "manual page" for each system command, organized alphabetically. Each manual page provides the following information:

- Command syntax including all switches and arguments
- Descriptions of all switches and arguments
- Description of what the command does (including hints on using the command)
- Examples of using the command
- List of related commands

Notational Conventions

This section describes the following notational conventions:

- Type style usage
- Command syntax descriptions

Type Style Usage

The text of this manual uses the following type style conventions:

Bold Identifies command names and switches, system call names, reserved words, and other items that must be used exactly as shown.

Italic Identifies variables, filenames, directories, partitions, and user names. Italic type style is also occasionally used to emphasize a word or phrase.

`Plain-Monospace`
Identifies computer output (prompts and messages), examples, and values of variables.

Bold-Italic-Monospace
Identifies user input (what you enter in response to some prompt).

`Bold-Monospace`
Identifies the names of keyboard keys (which are also enclosed in angle brackets). A dash indicates that the key preceding the dash is to be held down *while* the key following the dash is pressed. For example:

`<Break>` **`<s>`** **`<Ctrl-Alt-Del>`**

Command Syntax Descriptions

In this manual, the syntax of each system command is described in the "Syntax" section of the command's manual page. The following notational conventions apply to these syntax descriptions:

Bold Identifies command names and switches (i.e., items that you must use exactly as shown).

Italic Identifies arguments (that is, items whose values you must supply when you invoke the command).

[ ] (Brackets) Surround optional items (that is, items that can be omitted).

| (Vertical bar) Separates two or more items of which you may select *only* one.

{ } (Braces) Surround two or more items (separated by vertical bars) of which you *must* select only one.

For example, consider the syntax description of the **mkpart** command:

mkpart [**-sz** *size* | **-sz** *h*X*w* | **-nd** *nodespec*] [**-ss** | [[**-rq** *time*] [**-epl** *priority*]]] [**-mod** *mode*] *partition*

This syntax description shows the following:

- The *partition* argument is required. It is in italics because it is a name you can make up.
- You may choose one of **-sz** *size*, **-sz** *h*X*w*, or **-nd** *nodespec*.
- In addition, you may choose **-ss**. If you do not choose **-ss**, you may choose one or both of **-rq** *time* and **-epl** *priority*.
- Finally, you may choose **-mod** *mode*.

Applicable Documents

For more information, refer to the following manuals:

Paragon™ OSF/1 Manuals

- *Paragon™ OSF/1 User's Guide*
- *Paragon™ OSF/1 C System Calls Reference Manual*
- *Paragon™ OSF/1 Fortran System Calls Reference Manual*
- *Paragon™ OSF/1 C Compiler User's Guide*
- *Paragon™ OSF/1 Fortran Compiler User's Guide*

Intel® Manuals

- *i860™ 64-Bit Microprocessor Family Programmer's Reference Manual*

Other Manuals

- *OSF/1 User's Guide*
- *OSF/1 Programmer's Reference*
- *OSF/1 Command Reference*
- *OSF/1 System and Network Administrator's Reference*
- *OSF/1 Network Application Programmer's Guide*

Comments and Assistance

Intel Supercomputer Systems Division is eager to hear of your experiences with our products. Please call us if you need assistance, have questions, or otherwise want to comment on your Paragon system.

U.S.A./Canada Intel Corporation
phone: 800-421-2823
Internet: support@ssd.intel.com

Intel Corporation Italia s.p.a.
Milanofiori Palazzo
20090 Assago
Milano
Italy
1678 77203 (toll free)

France Intel Corporation
1 Rue Edison-BP303
78054 St. Quentin-en-Yvelines Cedex
France
0590 8602 (toll free)

Japan Intel Corporation K.K.
Supercomputer Systems Division
5-6 Tokodai, Tsukuba City
Ibaraki-Ken 300-26
Japan
0298-47-8904

United Kingdom Intel Corporation (UK) Ltd.
Supercomputer System Division
Pipers Way
Swindon SN3 IRJ
England
0800 212665 (toll free)
(44) 793 491056 (*answered in French*)
(44) 793 431062 (*answered in Italian*)
(44) 793 480874 (*answered in German*)
(44) 793 495108 (*answered in English*)

Germany Intel Semiconductor GmbH
Dornacher Strasse 1
8016 Feldkirchen bel Muenchen
Germany
0130 813741 (toll free)

World Headquarters
Intel Corporation
Supercomputer Systems Division
15201 N.W. Greenbrier Parkway
Beaverton, Oregon 97006
U.S.A.
(503) 629-7600

If you have comments about the Paragon manuals, please fill out and mail the enclosed Comment Card. You can also send your comments electronically to the following address:

techpubs@ssd.intel.com (Internet)

Table of Contents

application 1
ASYNC 8
BOOTMESH 10
BOOTPP 12
CBS 17
CHPART 19
FSPLIT 22
IPD 25
LSIZE 26
LSPART 28
MKPART 31
PARSEMAGIC 35
PMAKE 37
PSPART 50
RESET 52
RMPART 54
SAT 56
SCANIO 62
SHOWPART 65

application application

Controls the execution characteristics of a parallel application.

Syntax

application [**-sz** *size*] [**-pri** *priority*] [**-pt** *ptype*] [**-on** *nodespec*]
[**-pn** *partition*] [**-pkt** *packet_size*] [**-mbf** *memory_buffer*]
[**-mex** *memory_export*] [**-mea** *memory_each*] [**-sth** *send_threshold*]
[**-sct** *send_count*] [**-gth** *give_threshold*] [**-plk**] [*application_args*]
[**\;** *file* [**-pt** *ptype*] [**-on** *nodespec*] [*application_args*]] ...

Arguments

-sz *size* Sets the number of nodes to allocate to the application. The *size* argument must be an integer ranging from 1 to the number of nodes in the partition. The default is the value of the *NX_DFLT_SIZE* environment variable, or the number of nodes in the partition if the *NX_DFLT_SIZE* variable is not defined.

-pri *priority* Sets the application's priority level. The *priority* argument must be an integer ranging from 0 (low priority) to 10 (high priority). The default is a priority of 5.

-pt *ptype* Sets the process type of each process in the application. The *ptype* argument must be an integer greater than or equal to 0. The default is 0.

-on *nodespec* Loads the application on the nodes specified by the *nodespec* argument. The default loads the application on all the application's nodes. The *nodespec* must be one of the following node specifiers:

x	Node whose node number is *x*.
x..y	Range of nodes from numbers *x* to *y*, inclusive.
n	Letter **n** specifies the last node of the partition.
nspec[,*nspec*]...	List of nodes for the application. Each *nspec* argument is a node specifier. You can list nodes using any combination of the node specifiers *x*, *x..y*, or **n**. Do not put spaces in this list.

The numbers you use with the **-on** switch are node numbers within the application. The range of node numbers is from 0 to one less than the number of nodes allocated to the application.

-pn *partition* Specifies the partition in which the application runs. The *partition* argument must be the partition pathname of an existing partition. You need execute permission on the partition. The default partition is the value of the *NX_DFLT_PART* environment variable, or the *.compute* partition if the *NX_DFLT_PART* variable is not set.

NOTE

For the default, minimum, and maximum values of the message-passing configuration switches **-pkt**, **-mbf**, **-mex**, **-mea**, **-sth**, **-sct**, and **-gth**, the name *full_packet_size* represents the value *packet_size* + **sizeof(xmsg_t)**. The type **xmsg_t** is defined in the include file *<mcmsg/mcmsg_xmsg.h>*, and defines the message header sent with each packet. The size of this type is currently 32 bytes.

All values for the message-passing configuration switches are rounded down to the nearest multple of 32.

-pkt *packet_size*

Sets the packet size for sending messages (*packet_size*), in bytes. If a message is larger than the *packet_size* value, the application sends messages in several packets, each *packet_size* bytes long.

default: 1792 bytes or ((*memory_each* / 2) - **sizeof(xmsg_t)**) whichever is less
maximum: 1792
minimum: 32

-mbf *memory_buffer*

Sets the total memory allocated for message buffers in each process (*memory_buffer*), in bytes.

default: 1MB + (10 * (*full_packet_size*))
maximum: available physical memory
minimum: 2 * (2 * (*full_packet_size*) * (**numnodes**() + 2)) +
(10 * (*full_packet_size*)) for local messages

application *(cont.)* application *(cont.)*

-mex *memory_export*
Sets the total memory allocated for buffering messages received from other nodes (*memory_export*), in bytes.

default: *memory_buffer* - (10 * (*full_packet_size*))
maximum: *memory_buffer* - (10 * (*full_packet_size*))
minimum: 2 * (**numnodes**() + 2) * minimum *memory_each*

-mea *memory_each*
Sets the memory allocated to each node in the application for buffering messages received from other nodes (*memory_each*), in bytes. The application uses memory in the *memory_export* segment that is outside of **numnodes**() times the *memory_each* value for buffering messages from any sending node, when needed.

default: 10 * (*full_packet_size*) or maximum *memory_each* whichever is less
maximum: (*memory_export* / 2) / (**numnodes**() + 2)
minimum: 2 * (*full_packet_size*)

-sth *send_threshold*
Sets the threshold for sending multiple packets (*send_threshold*), in bytes. If a sending node has at least *send_threshold* bytes of memory free in its *memory_each* segment on the receiving node, it will send multiple packets of a message right away. Otherwise, it will send one packet and wait for an acknowledgment that a receive has been posted.

default: *memory_each* / 2
maximum: *memory_each* - 1
minimum: none

-sct *send_count* Sets the number of bytes to immediately send when the memory available is above the *send_threshold* value (*send_count*).

default: *memory_each* / 2
maximum: *memory_each*
minimum: *packet_size*

-gth *give_threshold*
Sets the threshold for "give me more messages" message, in bytes (*give_threshold*). A receiving node tells its senders how much free memory the sender has in its *memory_each* segment by "piggy-backing" information on other messages going to the sender. However, if there are no such messages, the sender can get out of date and stop sending messages because it thinks there is no free memory left for it on the receiver. If the receiver knows that the sender thinks it has less than *give_threshold* bytes of memory free, but there is really more memory available, it sends a special message to the sender telling it how much memory is really available.

default: *packet_size*
maximum: *memory_each*
minimum: *packet_size*

-plk Locks the entire data area of each process into memory, like the **plock()** function. See the *OSF/1 Programmer's Reference* for information on the **plock()** function. This switch also conditions message-passing code to run more efficiently by assuming that all data buffers are locked into memory. The default is not to lock.

application_args Additional arguments specific to the application.

file Loads *file* onto some or all of the same nodes as *application*. The *file* must be a compiled executable file. The **-pt** and **-on** switches following *file* specify the process type and nodes for *file*; the *application_args* following *file* specify additional application-specific arguments for *file*. All switches specified before the escaped semicolon (\;) apply both to the *application* and the *file*.

NOTE

The escaped semicolon (\;) before the *file* argument must be preceded and followed by a space or tab. Otherwise, it will be considered part of the preceding or following argument.

Description

The switches described in this manual page are available for applications linked with the **-nx** switch or for applications linked with the **-lnx** switch that call **nx_initve()**.

An application linked with the **-lnx** switch that calls the **nx_initve()** system call can override the command line switches.

application (cont.) *application (cont.)*

Examples

The following examples assume that *myprog*, *mymgr*, and *myworker* are parallel applications that were linked with the **-nx** switch.

To run the *myprog* application on all nodes of your default partition, enter the following:

```
myprog
```

To run the *myprog* application with a priority of 7 on 50 nodes of your default partition, enter the following:

```
myprog -pri 7 -sz 50
```

To allocate all the nodes of the *mypart* partition to the *mymgr* application, load the *mymgr* application onto node 0 of the partition with process type 1, and load *myworker* onto all nodes *but* node 0 with process type 0, enter the following:

```
mymgr -on 0 -pt 1 -pn mypart \; myworker -on 1..n
```

Errors

`Bad node specification`

You specified a node number that is greater than the largest node number in the partition with the **-nd** switch, or you used an improperly-formatted *nodespec* with the **-on** switch.

`Exceeds partition resources`

You specified an application size with **-sz** *size* that is greater than the partition size, or the *NX_DFLT_SIZE* environment variable specifies a size greater than the partition size. If you did not specify a partition with the **-pn** switch, check the size of the partition specified by *NX_DFLT_PART* environment variable.

`Give count invalid or out of range`

You specified a *give_threshold* argument with the **-gth** switch that is invalid or out of range.

`Invalid priority`

You specified a priority that is not between 0 (zero) and 10.

`Memory buffer invalid or out of range`

You specified a buffer size with the **-mbf** switch that is invalid or out of range.

`Memory each invalid or out of range`

You specified a buffer size with the **-mea** switch that is invalid or out of range.

`Memory export invalid or out of range`

You specified a buffer size with the **-mex** switch that is invalid or out of range.

`Packet size invalid or out of range`

You specified a packet size with the **-pkt** switch that is invalid or out of range.

`Partition not found`

You specified a partition with the **-pn** switch that does not exist. If you did not use the **-pn** switch, check the value of the *NX_DFLT_PART* environment variable.

`Partition permission denied`

You do not have execute permission for the partition. If you did not use the **-pn** switch, check the value of *NX_DFLT_PART*.

`Send threshold invalid or out of range`

You specified a send threshold with the **-sth** switch that is invalid or out of range.

`Send count invalid or out of range`

You specified a send count with the **-sct** switch that is invalid or out of range.

See Also

Commands: **lspart, mkpart, pspart, showpart**

Calls: **nx_initve()**

OSF/1 Programmer's Reference: **plock(2)**

ASYNC ASYNC

Connects a device for asynchronous communications between the diagnostic workstation and the Intel supercomputer.

Synopsis

async [**-t**] *device*

Arguments

-t	Returns `1` if the device is available (not locked), else returns `0`. The return value can be checked using the shell variable *$?*.
device	Name of the device to be used for asynchronous communications. The device name can be either an absolute pathname for a TTY device, such as */dev/ttyla*, or a simple device name. If you use a simple device name, the device is assumed to be in the */dev* directory.

Description

The **async** command is for use by the system administrator on the diagnostic station only.

The **async** command provides serial communications to the Intel supercomputer console with the following communication parameters: 19.2K baud, 8 data bits, 2 stop bits, and no parity. These communications parameters cannot be changed.

When the **async** command connects to a device, the command creates a file named */tmp/LOCK.XXX*. The *XXX* variable is the device name minus the directory. For example, when connected to */dev/ttyla*, the **async** command creates a file called */tmp/LOCK.ttyla*. This file acts like a lock so no other user can use the **async** command with this device. However, other commands such as **cu** can use the device causing unpredictable results.

The lock file contains the process ID (PID) of the process that owns the lock. The **async** command first checks to make sure the process still exists on the system. If the process does not exist, the lock is ignored.

ASYNC *(cont.)* ASYNC *(cont.)*

When the **async** command is running, entering a tilde (~) begins a command sequence. The following command sequences have special meaning:

~!	Execute the following operating system command. After the operating system command completes, you will be returned to the device. You can use the **sh** command to create a shell if you want to enter more than one command.
~. (dot)	Exit the program. Be careful using this when using **rlogin** to log in to the diagnostic station. Use two tildes and a dot (~~.) when logged in via the **rlogin** command.
~q	Exit the program. The ~q sequence is identical to ~. (dot). Use when logged in via the **rlogin** command to avoid killing the **rlogin** process.

These command sequences work only after entering a carriage return on the command line.

Bugs and Problems

If you attempt to open an invalid serial device, your process is killed.

Connecting to a serial device that is not attached to an Intel supercomputer, causes the connection to hang.

Files

/usr/local/bin/async	Contains the executable for the **async** command.
/tmp/LOCK.XXX	Lock file used to lock the device so no other users can use the **async** command on the device.

See Also

scanio

BOOTMESH BOOTMESH

Loads the Paragon™ OSF/1 operating system onto the mesh nodes.

Synopsis

bootmesh [**-bdDEGHKMpPRSvwzZ**] [**-e** *file*] [**-k** *file*] [**-m** *file*] [**-n** *node*] [**-s** *file*] [**-t** *millisecs*]

Arguments

-b	Broadcasts the operating system image to all nodes.
-d	Specifies maximum debug support.
-D	Specifies minimum debug support.
-e *file*	Specifies the emulator file to use for booting on the Intel supercomputer. The default is */mach_servers/emulator*.
-E	Does not download the emulator(s).
-G	Does not download the goto command(s).
-H	Displays help messages during execution.
-k *file*	Specifies the kernel file to use for booting on the Intel supercomputer. The default is */mach_servers/mach_kernel*.
-K	Does not download the kernel file.
-m *file*	Specifies the bootmagic master file to use for booting.
-M	Does not download the bootmagic file.
-n *node*	Specifies the first node to boot. The default is `0`.
-p	Does not poll for initialized servers.
-P	Polls all nodes despite a first timeout.
-R	Does not reset the NIC memory loader.
-s *file*	Specifies the server file for booting. The default is */mach_servers/startup*.

BOOTMESH *(cont.)* BOOTMESH *(cont.)*

-S	Does not download the server file.
-t *millisec*	Specifies timeout value for polling nodes.
-v	Displays all help messages.
-w	Suppresses displaying warning messages.
-z *seconds*	Specifies the number of seconds to sleep between goto commands. The *seconds* parameter is an integer value that specifies the number of seconds to sleep.
-Z	Specifies slow execution of the goto commands.

Description

The **bootmesh** command is for use by the system administrator on the diagnostic station only.

The **bootmesh** command runs during the booting sequence on the bootnode only. This command uses the information in the bootmagic file to find the Paragon OSF/1 operating system's microkernel, server, and emulator and download these files on the mesh nodes. When the downloading completes, this command start the microkernel on each mesh node.

Defaults

bootmagic file	The memory-resident bootmagic file.
configuration file	*/usr/paragon/boot/SYSCONFIG.TXT*
dskernel file	*/usr/paragon/boot/mach_kernel*
emulator file	*/mach_servers/emulator*
kernel file	*/mach_servers/mach_kernel*
master file	*/usr/paragon/boot/MAGIC.MASTER*
server file	*/mach_servers/startup*

See Also

bootpp, **parsemagic**

BOOTPP BOOTPP

Preprocesses information for the bootmagic file.

Synopsis

bootpp [**-dDFHvwZ**] [**-b** *file*] [**-c** *file*] [**-e** *file*] [**-E** *string*] [**-h** *number*] [**-I** *string*]
[**-k** *file*] [**-l** *node_list*] [**-m** *file*] [**-n** *node*] [**-p** *name=string*]
[**-P** *string*] [**-r** *device*] [**-s** *file*] [**-x** *size*] [**-y** *size*] [**-z** *file*]

Arguments

-b *file* Specifies the bootmagic file to use for booting. The default is */usr/paragon/boot/bootmagic*.

-c *file* Specifies the hardware configuration file to use for booting on the diagnostic station. The default is */usr/paragon/boot/SYSCONFIG.TXT*.

-d Specifies maximum debug support.

-D Specifies minimum debug support.

-e *file* Specifies the emulator file to use for booting on the Intel supercomputer. The default is */mach_servers/emulator*.

-E *string* Specifies the value for *EXPORT_PAGING*.

-F Creates a new master file for a new hardware or software configuration. This ignores the default or specified master file or configuration file. With this switch, you must specify configuration values with the **-l**, **-n**, **-x**, and **-y** switches. The values for these switches override the values in the installed hardware configuration file.

-h *number* Specifies a "howto" value for booting the Intel supercomputer. The *number* argument should be expressed in hexadecimal with the leading zero format (`0x` or `0X`). The */usr/include/sys/reboot.h* file on the Intel supercomputer defines the reboot values permitted for the *number* argument. The default value is `0x0`.

-H Displays help messages during execution.

BOOTPP *(cont.)* BOOTPP *(cont.)*

-I *string* Specifies the value for *IMPORT_PAGING*.

-k *file* Specifies the kernel file to use for booting on the Intel supercomputer. The default is */mach_servers/mach_kernel*.

-l *node_list* Specifies the node list to use for booting.

-m *file* Specifies the bootmagic master file to use for booting. The default is */usr/paragon/boot/MAGIC.MASTER*.

-n *node* Specifies the first node to boot. The default is `0`.

-p *name=string* Adds a boot parameter.

-P *string* Specifies the value for *PAGER_NODE*.

-r *device* Specifies the root device for booting. The default is `rz0a`.

-s *file* Specifies the server file for booting. The default is */mach_servers/startup*.

-v Displays all help messages.

-w Suppresses displaying warning messages.

-x *size* Specifies the number of nodes in the system's X dimension.

-y *size* Specifies the number of nodes in the system's Y dimension.

-z *file* Specifies doing a checksum on each downloaded operating system kernel.The *file* argument is the kernel file on the diagnostic station to use for the checksum. The default is */usr/paragon/boot/mach_kernel*.

-Z Suppresses the kernel checksum operation, but read access to the kernel file is validated. Both the checksum operation and file access validation are disabled if the **-F** switch is specified.

NOTE

The checksum operation compares a copy of the kernel file downloaded from the diagnostic node with the kernel file downloaded on each mesh node. If the checksum operation is enabled and the kernel downloaded from the diagnostic station does not match the kernel downloaded to the mesh nodes, booting cannot complete.

Description

The **bootpp** command is for use by the system administrator on the diagnostic station only.

The **bootpp** command runs in the reset script on the diagnostic station immediately before booting an Intel supercomputer. It prepares the bootmagic file, which communicates hardware and software configuration information to the Paragon™ OSF/1 operating system.

The bootmagic file consists of a set of strings with the following form:

```
name=value
```

The string values are terminated with a new line and the bootmagic file is terminated with a null character.

The bootmagic file contains the value that initializes the time-of-day clock for the Intel supercomputer. For this reason, boot the Intel supercomputer system immediately after running the **bootpp** command.

A # character at the beginning of a line in the *MASTER.MAGIC* or *SYSCONFIG.TXT* file indicates the line is a comment line.

Generating Bootmagic Strings

The **bootpp** command uses the following levels of information to prepare the bootmagic file:

1. Default configuration parameters used in the absence of any other inputs.

2. Master file containing default bootmagic strings is consulted if it is present. Values in this file override the default configuration parameters. The default master file has the pathname */usr/paragon/boot/MAGIC.MASTER* or the pathname you specify with the **-M** switch.

3. A hardware configuration file (*SYSCONFIG.TXT* or the pathname specified with the **-C** switch) containing a description of the actual operational Intel supercomputer hardware is consulted if it is present. Values in this file override the master file and default values related to node configuration (such as, `mesh_x`, `mesh_y`, `bootnode`, `node_list`).

4. Command line switches override the values set by defaults or set in the master file or configuration file for a configuration parameter.

5. Strings for new configuration parameters can be specified on the **bootpp** command line with the **-p** switch or inserted into the bootmagic file with the following form:

```
name=value
```

The following cases are exceptions to the rules for bootmagic strings:

1. When relying on the defaults to generate a bootmagic file, you still must use the **bootpp** command to specify the hardware mesh configuration parameters using the **-l**, **-n**, **-x**, and **-y** switches. There are no defaults for the values of these switches.

2. The pathnames of the bootmagic file, the master file, and the hardware configuration file can only be specified using the defaults or command line arguments. For example, you cannot specify the master file pathname in the bootmagic file.

3. The list of operational nodes in the root partition (`node_list`) is always computed from the available information about operational nodes.

4. The time-of-day value is always determined by a direct query (**time(3)**) to the diagnostic station.

Defaults

bootmagic file	*/usr/paragon/boot/bootmagic*
bootnode	`0`
bootpp	*/usr/local/bin/bootpp*
configuration file	*/usr/paragon/boot/SYSCONFIG.TXT*
dskernel file	*/usr/paragon/boot/mach_kernel*
emulator file	*/mach_servers/emulator*
howto	`0x0`
kernel file	*/mach_servers/mach_kernel*
master file	*/usr/paragon/boot/MAGIC.MASTER*
root device	*rz0a*
server file	*/mach_servers/startup*

BOOTPP *(cont.)*

BOOTPP *(cont.)*

See Also

cbs, parsemagic

CBS CBS

Specifies the cabinet, backplane, slot node numbering system for the Intel supercomputer.

Description

CBS is a node numbering system for the Intel supercomputer.

Cabinet numbering starts on the cabinet's front right with 00 and increments across to the cabinet's front left.

Backplane numbering starts with the letter A at the bottom of the cabinet and increments to the letter D at the top of the cabinet. Slot numbering starts with the number 0 (zero) on the right side of each backplane and goes to the number 15.

A node number combines the two-digit cabinet number, the one-character backplane name, and the two-digit slot number.

For example, in the following diagram the CBS number indicated by the two stars (**) is specified as `01B10`.

```
   +-----------+-----------+
D  | 12 11 4 3 | 12 11 4 3 |
   | 13 10 5 2 | 13 10 5 2 |
   | 14  9 6 1 | 14  9 6 1 |
   | 15  8 7 0 | 15  8 7 0 |
   +-----------+-----------+
C  | 12 11 4 3 | 12 11 4 3 |
   | 13 10 5 2 | 13 10 5 2 |
   | 14  9 6 1 | 14  9 6 1 |
   | 15  8 7 0 | 15  8 7 0 |
   +-----------+-----------+
B  | 12 11 4 3 | 12 11 4 3 |
   | 13 ** 5 2 | 13 10 5 2 |
   | 14  9 6 1 | 14  9 6 1 |
   | 15  8 7 0 | 15  8 7 0 |
   +-----------+-----------+
A  | 12 11 4 3 | 12 11 4 3 |
   | 13 10 5 2 | 13 10 5 2 |
   | 14  9 6 1 | 14  9 6 1 |
   | 15  8 7 0 | 15  8 7 0 |
   +-----------+-----------+
         01          00
```

CBS *(cont.)*

CBS *(cont.)*

See Also

bootpp, parsemagic

CHPART CHPART

Changes a partition's characteristics.

Syntax

chpart [**-epl** *priority*] [**-g** *group*] [**-mod** *mode*] [**-nm** *name*] [**-o** *owner*[. *group*]] [**-rq** *time*] *partition*

Arguments

-epl *priority* Changes the partition's effective priority limit to *priority*, where *priority* is an integer from 0 to 10 inclusive. The **-epl** argument can be used only on a gang-scheduled partition.

-g *group* Changes the partition's group. The *group* argument can be either a group name or number. You must be a member of the group that currently owns the partition, and you must be a member of the new group for the partition.

-mod *mode* Specifies the partition's protection modes. The *mode* value can be specified as a three-digit octal number with the form *nnn* (see the **chmod** command) or a nine-character string with the form `rwxrwxrwx`, where a letter (`r`, `w`, or `x`) represents a permission granted and a dash (`-`) represents a permission denied (see the **ls** command's **-l** switch). You must own the partition to use this switch. See the *OSF/1 Command Reference* for more information about the **chmod** and **ls** commands.

-nm *name* Changes the partition's name. This switch changes the partition's name only. You cannot use this switch to change the partition's parent partition or the partition's relationship in a partition hierarchy. The *name* argument must be a simple name (without dots).

-o *owner*[. *group*]
Changes the partition's owner to *owner*. If the . *group* argurment is specified, this also changes the partition's group to the value of the *group* argument. The *owner* and *group* values can be either user/group names or numeric user/group IDs. You must be *root* user to use this switch.

CHPART *(cont.)* CHPART *(cont.)*

-rq *time* Changes the partition's rollin quantum to *time*, where *time* is one of the following:

n	*n* milliseconds (if *n* is not a multiple of 100, it is rounded up to the next multiple of 100).
*n***s**	*n* seconds.
*n***m**	*n* minutes.
*n***h**	*n* hours.
0	"Infinite" time: once rolled in, an application runs until it exits.

The *time* cannot be greater than 24 hours. The **-rq** argument can be used only on a gang-scheduled partition.

partition Absolute or relative pathname of a partition.

Description

The **chpart** command lets you change the following partition characteristics: rollin quantum, effective priority limit, protection modes, owner, and group.

You must have write permission on the specified partition to use this command. Additional permissions are required for the **-g**, **-mod**, and the **-o** switches. See the descriptions of these switches.

Examples

The following command changes the rollin quantum of *mypart* to 20 minutes:

```
chpart -rq 20m mypart
```

The following command changes the effective priority limit of *mypart* to 2:

```
chpart -epl 2 mypart
```

The following command changes the protection modes of *mypart* so that it is readable, writable, and executable by the owner but not by anyone else:

```
chpart -mod rwx------ mypart
```

CHPART *(cont.)* CHPART *(cont.)*

The following command changes the owner of *mypart* to *smith*, but does not affect the group:

```
chpart -o smith mypart
```

Errors

`chpart: Invalid partition rename`

You specified how a partition name that was not a simple name.

`chpart: Partition lock denied`

You specified a partition that is currently in use and being updated by someone else. You cannot change the characteristics of a partition that is currently being updated.

`chpart: Partition not found`

You specified a partition that does not exist.

`chpart: Partition permission denied`

You specified a partition for which you do not have write permission, or you tried to change the owner of the partition when you are not root. You specified the **-g** switch when you are not in the group specified by the *group* parameter. You specified the **-mod** switch when you are not the owner of the partition.

See Also

Commands: **lspart**, **mkpart**, **pspart**, **rmpart**, **showpart**

Calls: **nx_chpart()**

FSPLIT FSPLIT

Splits one file containing several Fortran program units into several files containing one program unit each.

Syntax

fsplit [*filename*]

Arguments

filename	Identifies the file that you want to split into individual files. If you omit this argument, **fsplit** reads standard input.

Description

Use **fsplit** to split a single file containing several Fortran program units (program, subroutine, function, or block data) into several files, each of which contains only one program unit. The first non-comment line of each program unit (usually a **program**, **subroutine**, **function**, or **block data** statement) marks the beginning of each kind of program unit; the **end** statement marks the end.

The original file remains unchanged; the new files are named as follows:

- Each named program unit (one that specifies a name in its **program**, **subroutine**, **function**, or **block data** statement) is put in a file named *name*`.f`, where *name* is the name of the program unit.
- An unnamed block data subprogram is put in a file named `BLOCKDATAn.f`, where *n* is a number that corresponds to the *n*th unnamed block data subprogram in the original file.
- A main program that does not contain a **program** statement is put in a file named `MAIN.f`.

WARNING

If a file of the same name already exists in the current directory, **fsplit** silently overwrites it. This means that if your file defines multiple program units with the same name, the resulting `name.f` file will contain only the last program unit defined. In addition, if your file has several program units that do not have a **program**, **subroutine**, **function**, or **block data** statement, the `MAIN.f` file will contain only the last one. Be careful.

Example

Consider the following source file (named *file.f*):

```
      program a
c  This is the first main program named a.
      end

      program a
c  This is the second main program named a.
      end

      integer a-m
c  This is the first unnamed main program unit.
      end

c  This is the second unnamed main program unit.
      integer n-z
      end

      subroutine b
c  This is the first subroutine named b.
      end

      subroutine b
c  This is the second subroutine named b.
      end

c  This is the function named c.
      function c
      end

      block data d
c  This is the block data program unit named d.
      end

      block data
c  This is the first unnamed block data program unit.
      end

      block data
c  This is the second unnamed block data program unit.
      end
```

FSPLIT *(cont.)* FSPLIT *(cont.)*

Before using **fsplit**, the directory contains the following:

```
ls
file.f
```

After using **fsplit**, the directory contains the following:

```
fsplit file.f
ls
BLOCKDATA1.f    MAIN.f          b.f             d.f
BLOCKDATA2.f    a.f             c.f             file.f
```

Examining the various files reveals the following:

```
cat BLOCKDATA1.f
      block data
c  This is the first unnamed block data program unit.
      end
cat BLOCKDATA2.f
      block data
c  This is the second unnamed block data program unit.
      end
cat MAIN.f
      integer n-z
      end
cat a.f
      program a
c  This is the second main program named a.
      end
cat b.f
      subroutine b
c  This is the second subroutine named b.
      end
cat c.f
      function c
      end
cat d.f
      block data d
c  This is the block data program unit named d.
      end
```

Note that the files *a.f*, *b.f*, and *MAIN.f* contain the second program units defined in the original source file. In addition, the files *MAIN.f* and *c.f* do not contain the comments that were associated with each in the original source file. This example highlights some of the potential "gotchas" in using the **fsplit** command.

IPD IPD

Invoke the Interactive Parallel Debugger.

Syntax

ipd

Description

The **ipd** command invokes the Interactive Parallel Debugger (IPD). IPD is a complete symbolic, source-level debugger for parallel programs that run under the Paragon™ OSF/1 operating system. Beyond the standard operations that facilitate the debugging of serial programs, IPD offers custom features that facilitate debugging parallel programs. IPD lets you debug parallel programs written in C, Fortran, and assembly language.

IPD consists of a set of debugging commands, for which help is available from within IPD. After invoking IPD, entering either **help** or **?** at the IPD prompt returns a summary of all IPD commands.

See Also

Paragon™ OSF/1 Interactive Parallel Debugger Manual

LSIZE LSIZE

Sets or increases the size of one or more files.

Syntax

lsize [**-a**] *size file ...*

Arguments

-a Increases the file size by *size* bytes. If you do not specify the **-a** switch, **lsize** changes the file size to be exactly *size* bytes.

size Number of bytes used to set or increase the file size. To specify units other than bytes, append the appropriate letter to the *size* argument:

k	Kilobytes (1024 bytes)
m	Megabytes (1024K bytes)
g	Gigabytes (1024M bytes)

file Pathname of the file to be changed. If it exists, you must have write permission on this file. If it does not exist, you must have write permission on the file's directory.

Description

Use the **lsize** command to increase or set the disk space allocated for one or more files. There are two forms of this command. The following form sets the file size to *size* bytes:

lsize *size file ...*

The second form increases the file size by *size* bytes:

lsize -a *size file ...*

If a specified file does not exist, this command creates the file and allocates *size* bytes for the file.

LSIZE *(cont.)* LSIZE *(cont.)*

The **lsize** command cannot decrease the size of a file. If you specify a file size smaller than the file's current size, the command has no effect on the file.

The **lsize** command allocates file space for the file from the file system's disk space. The file's new file space is not initialized; the new file space contains whatever existed in the disk space the last time it was used.

Examples

The following command sets the size of the file *mydata* to 5M bytes:

```
lsize 5m mydata
```

The following command increases the size of the file *mydata* by 200K bytes:

```
lsize -a 200k mydata
```

Errors

`lsize: No space left on device, n bytes allocated for <file>.`

You specified a size to allocate for the file that exceeds the space available on the device.

`lsize: Invalid size.`

You specified a size that has an invalid format.

`lsize: Size size is too large.`

You specified a size that exceeds the size of an integer.

See Also

Calls: **esize()**, **lsize()**

LSPART LSPART

List information associated with subpartitions of a partition.

Syntax

lspart [**-r**] [*partition*]

Arguments

-r Recursively lists information about all the subpartitions of the specified partition.

partition Absolute or relative partition pathname of the partition to be listed. If you do not specify a partition argument, the **lspart** command uses the value specified in *NX_DFLT_PART*. If *NX_DFLT_PART* is not set, the default is *.compute*.

Description

Use the **lspart** command to list the characteristics of the subpartitions of a partition. You must have read permission on the specified partition to list its subpartition information.

The **lspart** command shows the following information about the subpartitions of a partition:

`USER` The login name of the user who owns the subpartition.

`GROUP` The group name of the group that owns the subpartition.

`ACCESS` The access permissions for the subpartition.

`SIZE` The number of nodes allocated to the subpartition. If two node numbers are separated by a slash (/), it indicates that one or more of the nodes for the indicated partition are unusable.

`RQ` The subpartition's rollin quantum.

`EPL` The subpartition's effective priority limit.

`PARTITION` The name of the subpartition.

The values for the SIZE, ROLLIN, and EPL characteristics are shown as an asterisk (*) if you do not have read permission on the partition or subpartition, and a dash (-) if these characteristics are not set.

LSPART *(cont.)* LSPART *(cont.)*

Examples

To list the subpartitions of the partition called *mypart*, whose parent partition is the compute partition, you can use the following command:

```
lspart mypart

USER    GROUP   ACCESS  SIZE    RQ      EPL     PARTITION
chris   eng     777     16      15m     3       mandlebrot
pat     mrkt    755      4      10m     10      slalom
```

In this case, *mypart* has two subpartitions: *mandelbrot* and *slalom*. *mandelbrot* is owned by user *chris* in group *eng*; it has permissions `rwxrwxrwx`, a size of 16 nodes, a rollin quantum of 15 minutes, and an effective priority limit of 3. The *slalom* subpartition is owned by user *pat* in group *mrkt*; it has permissions `rwxr-xr-x`, a size of 4 nodes, a rollin quantum of 10 minutes, and an effective priority limit of 10.

The following command has the same effect, but uses an absolute partition pathname:

```
lspart .compute.mypart
```

To recursively list all of a partition's subpartitions, sub-subpartitions, and so on, use the **-r** switch. For example:

```
lspart -r mypart

  USER    GROUP   ACCESS  SIZE    RQ      EPL     PARTITION
mypart:
  chris   eng     777     16      15m     3       mandelbrot
  pat     mrkt    755      4      10m     10      slalom
mypart.mandelbrot:
  chris   eng     777     16      15m     10      hi_pri
  chris   eng     777     16      15m     1       lo_pri
```

The **lspart -r** output reveals that *mypart.mandelbrot* has two subpartitions, *hi_pri* and *lo_pri*, neither of which has any sub-subpartitions; *slalom* has no subpartitions.

LSPART *(cont.)* LSPART *(cont.)*

The following example shows that one or more of the nodes for the indicated partition are unusable. For example:

```
lspart mypart

USER    GROUP   ACCESS  SIZE      RQ   EPL    PARTITION
chris   eng     777     14 / 16   15m  3      mandlebrot
```

In this example, `14 / 16` shows there are 16 nodes allocated to the *mandelbrot* partition, but only 14 nodes are usable and 2 nodes are unusable. You cannot run applications or allocate any partitions on unusable nodes. Use the **showpart** command to find the specific unusable nodes in a partition.

Errors

`lspart: Partition not found`

You specified a partition that does not exist.

`lspart: Partition permission denied`

You do not have read permission for the specified partition.

See Also

chpart, mkpart, pspart, rmpart, showpart

MKPART MKPART

Makes a new partition.

Syntax

mkpart [**-sz** *size* | **-sz** *h*X*w* | **-nd** *nodespec*] [**-ss** | [[**-rq** *time*] [**-epl** *priority*]]]
[**-mod** *mode*] *partition*

Arguments

-sz *size* Specifies the size of the partition you are making. The *size* argument must be an integer between 0 and the number of nodes in the parent partition. The default is all the usable nodes of the parent partition.

-sz *h*X*w* Creates a contiguous rectangular partition that is *h* nodes high and *w* nodes wide. (You can use an uppercase or lowercase letter **X** between the integers *h* and *w*.) The default is all the usable nodes of the parent partition.

-nd *nodespec* Creates a partition that consists of exactly the specified nodes, where *nodespec* is one of the following:

x	The node whose node number is *x*.
x..y	The range of nodes from numbers *x* to *y*.
*h*X*w*:*n*	The rectangular group of nodes that is *h* nodes high and *w* nodes wide and whose upper left corner is node number *n*. (You can use an uppercase or lowercase letter **X** between the integers *h* and *w*.)
nspec[,*nspec*]...	The specified list of nodes, where each *nspec* is a node specifier of the form *x*, *x..y*, or *h*X*w*:*n*. Do not put any spaces in this list.

The numbers you use with **-nd** are node numbers within the parent partition, which always range from 0 to one less than the size of the partition. The default is all the usable nodes of the parent partition.

-ss Creates a partition that uses standard scheduling. This switch cannot be used with the **-rq** or **-epl** switches. If you don't use the **-ss**, **-rq**, or **-epl** switch, the new partition uses the same scheduling technique, rollin quantum, and effective priority limit as its parent partition.

MKPART *(cont.)*

MKPART *(cont.)*

-rq *time* Creates a partition that uses gang scheduling with a rollin quantum of *time*, where *time* is one of the following:

n	*n* milliseconds (if *n* is not a multiple of 100, it is silently rounded up to the next multiple of 100).
*n***s**	*n* seconds.
*n***m**	*n* minutes.
*n***h**	*n* hours.
0	"Infinite" time: once rolled in, an application runs until it exits.

The rollin quantum cannot be more than 24 hours.

The **-rq** argument can be used with or without **-epl**. If you use **-rq** without **-epl**, the new partition has the same effective priority limit as its parent partition.

If you don't use the **-ss**, **-rq**, or **-epl** switch, the new partition uses the same scheduling technique, rollin quantum, and effective priority limit as its parent partition.

-epl *priority* Creates a partition that uses gang scheduling with an effective priority limit of *priority*, where priority is an integer from 0 to 10 inclusive (0 is low priority, 10 is high priority).

The **-epl** argument can be used with or without **-rq**. If you use **-epl** without **-rq**, the new partition has the same rollin quantum as its parent partition.

If you don't use the **-ss**, **-rq**, or **-epl** switch, the new partition uses the same scheduling technique, rollin quantum, and effective priority limit as its parent partition.

-mod *mode* Specifies the partition's protection modes. The *mode* value can be specified as a three-digit octal number with the form *nnn* (see the **chmod** command) or a nine-character string with the form `rwxrwxrwx`, where a letter (`r`, `w`, or `x`) represents a permission granted and a dash (`-`) represents a permission denied (see the **ls** command with the **-l** switch). See the *OSF/1 Command Reference* for more information about the **chmod** and **ls** commands.

partition The absolute or relative partition pathname of the partition to be created.

MKPART *(cont.)* MKPART *(cont.)*

Description

Use the **mkpart** command to create a partition. The specified partition cannot already exist, and the parent partition must exist and give you write permission.

When you create a partition, you become the new partition's owner and the new partition's group is set to your current group.

The mkpart command let's you specify most of the partitions's characteristics. By default, the partition you are creating gets the same characteristics as the parent partition. You use the **mkpart** command's switches to set specific characteristics for a partition.

If you don't use the **-sz** or **-nd** switch, all the nodes of the parent partition are allocated to the new partition. If the parent partition is the root partition and it contains unusable nodes, only the usable nodes are allocated. You can use at most one **-sz** or **-nd** switch in a single **mkpart** command.

No matter how you specify the partition's size, nodes are always numbered from 0 to one less than the partition's size. In most cases, they are numbered from left to right and then top to bottom as they are located in the partition. If you use the **-nd** switch, the nodes in the new partition are numbered in the order you specify them in the **-nd** switch.

Examples

To create a partition called *mypart* whose parent partition is the compute partition, you can use the following command:

```
mkpart mypart
```

The following command has the same effect, but uses an absolute partition pathname:

```
mkpart .compute.mypart
```

Errors

`mkpart: Partition permission denied`

You specified a partition in a parent partition that does not give you write permission.

`mkpart: Exceeds partition resources`

You specified a partition size with **-sz** *size* that is greater than the number of nodes in the parent partition, or you specified a rectangle with **-sz** *h***X***w* that does not fit in the largest contiguous rectangle of nodes within the parent partition.

`mkpart: Bad node specification`

You specified a node number with **-nd** that is greater than the largest node number in the partition, or you specified an improperly-formatted *nodespec* with **-nd**.

`mkpart: Invalid priority`

You specified a partition whose priority is not between 0 (zero) and 10.

See Also

chpart, lspart, pspart, rmpart, showpart, ***application***

OSF/1 Programmer's Reference: **chmod(1), ls(1)**

PARSEMAGIC PARSEMAGIC

Returns values from the bootmagic file on the diagnostic station.

Synopsis

parsemagic [**-dlLnstv**] [**-B** *file*] [**-b** [*letter*]] [**-c** [*config*]] [**-m** [*mode*]]

Arguments

-B *file* Specifies the bootmagic file's pathname. The default is the bootmagic file in the current directory.

-b [*letter*] Specifies the backplane to use for a condo configuration. The *letter* argument is the cabinet name. The letter argument can be **A**, **B**, **C**, or **D**. The default is for the root nodes to be numbered relative to the top left node in the last cabinet. The default is **D**.

-c [*config*] Specifies the system configuration. The values for the *config* argument are `condo`, `full`, and `multi`. The default is `full`.

-d Displays a description of the system.

-l Returns a list of nodes with ranges expanded. This means node specifications in the format `3 . . 7` are displayed as `3 4 5 6 7`.

-L Returns a list of mesh routing chips (MRCs) with ranges expanded. This means MRC specifications in the format `3 . . 7` are displayed as `3 4 5 6 7`.

-m [*mode*] Used by the **reset** script to get information about the MRCs in a backplane. The *mode* parameter can be one of the following:

`fast`	Fast streaming mode
`slow`	Slow streaming mode
`interlocked`	Interlocked mode

Only the first character of the *mode* parameter is examined.

-n Displays the number of cabinets in the system.

-s Displays the slot number of the boot node.

PARSEMAGIC *(cont.)* PARSEMAGIC *(cont.)*

-t Tests the *BOOT_CONSOLE* string to see if it supports the scan interface. Returns `1` if it does support the scan interface, otherwise it returns `0` (zero) if it does not.

-v Verifies the existence of the bootmagic file. Returns `1` if the file exists and is readable. Returns `0` if the file is missing or is unreadable.

Description

The **parsemagic** command is for use by the system administrator on the diagnostic station only.

The **parsemagic** command reads values from the bootmagic file and returns the values to a shell program. This command fails if there is no bootmagic file in the current directory or if you do not specify a bootmagic file pathname.

Files

/usr/local/bin/parsemagic	Contains executable for the **parsemagic** command.

See Also

cbs, reset

PMAKE PMAKE

Parallel make utility that maintains up-to-date versions of target files and performs shell programs in parallel.

Syntax

pmake [**-bcdeFikmnNpqrsStuUvw**] [**-C** *dir*] [**-f***file*] [**-I** *dir*] [**-j** [*jobs*]] [**-l** [*load*]] [**-o** *file*] [**-P** *partition*] [**-W** *file*] [*macro_definition ...*] [*target ...*]

Arguments

-b Has no effect; exists so older-version **make** dependency files continue to work.

-c Does not try to find a corresponding Revision Control System (RCS) or Source Code Control System (SCCS) file and check it out if the file does not exist.

-C *dir* Changes to directory *dir* before reading the description files or doing anything else. If multiple **-C** switches are specified, each is interpreted relative to the previous one: **-C/ -C**`etc` is equivalent to **-C** `/etc`. This is typically used with recursive invocations of **pmake**.

-d Prints debugging information in addition to normal processing. The debugging information includes information about the files considered for processing, the comparison of file-times, the files that need processing, and the implicit rules being considered and actually applied.

-e Does not reassign environment variables within the description file.

-f *file* Reads *file* for a description of how to build the target file. If you do not specify the **-f** switch, **pmake** looks in the current directory for a description file named *makefile* or *Makefile*. If a **-** (dash) follows the **-f** switch, **pmake** reads standard input. You can specify more than one description file by entering the **-f** flag more than once (with its associated *file* argument).

-F Causes a fatal error if a description file is not present.

-i Ignores error codes returned by commands and continues to execute until finished. This is similar to the pseudotarget command **.IGNORE:**, which can be specified in the description file. The **pmake** command normally stops if a command returns a nonzero code.

PMAKE *(cont.)* PMAKE *(cont.)*

-I *dir* Specifies a directory *dir* to search for description files to be included. You can specify the **-I** switch multiple times in a command line to specify multiple directories to search. The directories are searched in the order specified.

-j [*jobs*] Specifies the maximum number of jobs that can run simultaneously. The default is the partition size, or 1 if the **pmake** command is running in the service partition. If there is more than one **-j** switch, the last one is effective. If the **-j** switch is given without an argument, the **pmake** command does not limit the number of jobs that can run simultaneously.

-k Stops processing the current target if an error occurs, but continues with other branches that do not depend on the target that failed.

-l [*load*] Specifies that no new jobs should be started if there are other jobs running and the load average is at least the value of *load* (a floating-point number). Specifying the switch with no argument removes a previous load limit.

-m Searches for machine-specific subdirectories automatically. On a Paragon system, if a *PARAGON* subdirectory exists in the current directory, the **-m** switch adds the *PARAGON* subdirectory to the directory list specified by the *VPATH* special variable. See the "Special Variables" section for more information.

-n Echoes commands that would be executed, but does not execute them.

-N Disables all configuration file (*Makeconf*) processing.

-o *file* Does not process *file* even if it is older than its dependencies, and does not process anything because of changes in *file*. Essentially, the file is treated as very old and its rules are ignored.

-p Echoes all the environment variables, macro definitions, and target descriptions before executing any commands. This also prints the version information given by the **-v** switch. To print the database without trying to remake any files, use the following:

```
pmake -p -f /dev/null
```

-P *partition* Run the **pmake** command as a parallel application in the partition specified by *partition*. The default is the service partition.

-q Does not execute the commands in the description file. Returns a status code of zero if the object files are up-to-date; otherwise, returns a nonzero value.

PMAKE *(cont.)* PMAKE *(cont.)*

-r Eliminates the built-in implicit rules and clears out the default list of suffixes for suffix rules.

-s Does not echo the commands being executed. This is similar to the pseudotarget command **.SILENT:**, which would be specified in the description file.

-S Stops processing the current target if an error occurs and does not continue to any other branch. This is the default. This cancels the effect of the **-k** switch. This is not necessary except in a recursive **pmake** where **-k** might be inherited from the top-level **pmake** via *MAKEFLAGS* or if you set **-k** in *MAKEFLAGS* in your environment.

-t Touches the targets; marks the files up-to-date without running commands to update them, or creates the target if it does not exist.

-u Does not unlink files that were automatically checked out from SCCS or RCS.

-U Has no effect; exists so older-version **make** dependency files continue to work.

-v Prints the version of the **pmake** command, a copyright, a list of authors, and a notice that there is no warranty. After this information is printed, processing continues normally. To get this information without doing anything else, use the following:

```
pmake -v -f /dev/null
```

-w Prints a message containing the working directory before and after other processing in the directory. This may be useful for tracking down errors from complicated nests of recursive **pmake** commands.

-W *file* Pretends that the target *file* has just been modified. When used with the **-n** switch, this shows you what would happen if you were to modify that file. Without **-n**, it is almost the same as running a *touch* command on the given file before running the **pmake** command, except that the modification time is changed only in the imagination of the **pmake** command.

macro_definition

Specifies a macro to use with the definition file. Use the same macro syntax as required for the definition file. Enclose strings in quotes. Spaces and tabs are ignored. See the "Macros" section for more information on using macros.

target Name of the target to build or update. Target names are typically executable files, but this is not always the case. If *target* is not specified, **pmake** uses the first target in the definition file.

Description

The **pmake** command updates a target based on whether the target's dependencies have been modified relative to the last modification of the target. Targets are typically executable files, but this is not always the case. Typically you use **pmake** to recompile large programs, but you can use it for any applications where a target must be updated whenever files the target depends on change. The **pmake** command executes commands in a description file, also called a *makefile*, to build or update the target.

The **pmake** command is an extension of GNU **make**, which was written by Richard Stallman and Roland McGrath. It has been renamed **pmake** to emphasize its ability to execute several commands in parallel, and to distinguish it from the standard sequential **make** utility. This manual page provides a summary of the base GNU **make** features as well as the **pmake** extensions. For more detailed information about GNU **make** features, refer to *The GNU Make Manual*. This manual is available through the Intel Supercomputer Systems Division Customer Service Response Center (*support@ssd.intel.com*).

For switches that have arguments, spaces between switches and argument are optional. For example, when using the **-I** switch, the directory name may come directly after the switch (**-I***dir*) or a space may be inserted (**-I** *dir*).

Description Files

When a description file exists for a program, invoking **pmake** executes all the commands needed to build the program. The **pmake** command uses a definition file and the last-modification time of the target and the files that a target depends on to decide which targets need updating.

Description files contain a sequence of entries that define target names and dependencies and describe the rules for updating the targets. A typical entry includes a dependency line and a series of commands, and takes the following form:

target1 [*target2* ...] :[:][*dependency1* ...] [; *command*]
[*command*] [; *command* ...]

The dependency line begins with one or more target names separated by spaces. A single or double colon separates the target(s) from a list of zero or more dependencies for the target(s). If no dependencies are given, the target files are always updated if they do not exist. Otherwise, a target is updated only if a dependency has changed since the target was last updated. A single command may also appear on the dependency line, separated from the dependencies by a semicolon. Alternatively, commands may appear on subsequent lines provided that each command line begins with a tab character. When a target requires updating, the specified commands are executed.

Dependency Lines

Dependency lines may take several forms:

target... : [*dependency*] ...
: Single-colon rules. Words following the colon are added to the dependency list for the target(s). If a target is named in more than one single-colon rule, the dependencies for all of its entries are concatenated to form that target's complete dependency list. In that case, only one of the single colon rules may include commands for remaking the target.

target... :: [*dependency*] ...
: Double-colon rules. When used in place of a single colon (:), the double colon (::) allows a target to be checked and updated with respect to alternate dependency lists. Each double colon rule is considered independently when deciding whether and how to update a particular target.

target... : *target-pattern* : *dep-pattern...*
: Static-pattern rules. The *target-pattern* and *dep-pattern* values specify how to compute the dependencies for each target. Each target is matched against the *target-pattern* to extract a part of the target name, called the stem. The stem is then substituted into each of the *dep-pattern* values to make the dependency names, for example, the following dependency line specifies that *foo.c* and *foo.h* are dependencies of *foo.o*:

```
$(OBJECTS): %.o : %.c %.h
```

.s1.s2 :
: Double-suffix rules. The rule tells how to make a file *foo.s2* from the file *foo.s1* where *foo* is an arbitrary stem and *s1* and *s2* are suffixes.

.s1 :
: Single-suffix rules. The rule tells how to make a file *foo* from the file *foo.s1* where *foo* is an arbitrary stem and *s1* is a suffix.

target-pattern : *dependency-pattern...*
: Pattern rules. The target and dependency patterns each contain the wild card character, % (percent). The % in the *target-pattern* is matched against a stem of a target name. That stem is substituted for the % in the *dependency-pattern*. This creates the dependency for the target, for example, %.o : %.c with the stem *foo* creates the target *foo.o* from the dependency *foo.c*.

Parallel Execution

The **pmake** command updates multiple target files in parallel. Parallel execution may occur either in the service partition or the compute partition. In the service partition, **pmake** relies on process migration and load balancing to ensure parallel execution. In the compute partition, **pmake** places commands on the available nodes within a partition and executes as a parallel application.

You can request parallel execution using either the **-j** or **-P** switch, or using both switches together. The **-P** switch specifies the partition in which **pmake** runs jobs. The default is the service partition. The **-j** switch specifies the maximum number of jobs that can run in parallel. If the **-j** switch is not used, the maximum number of jobs defaults to the number of nodes in the partition, or one job if the **pmake** command is running in the service partition. If you use the **-j** switch followed by the optional *jobs* argument, the **pmake** command runs up to the number of jobs you specify in parallel. The number of jobs the **pmake** command can run in parallel is not limited to the number of nodes in the partition., because multiple jobs can run on a node. If you specify the **-j** switch without the *jobs* argument, the maximum number of jobs the **pmake** command can run in parallel is unlimited.

The **-l** switch also affects parallel execution of the **pmake** command. This switch specifies that no new jobs should be started if there are other jobs running and the load average is at least the value of the *load* argument.This switch can limit parallel execution when the system load level is above the specified load level. The **-l** switch does not itself cause parallel execution. It affects parallel execution when you use it with the **-j** or the **-P** switches, only.

The **pmake** command relies on the dependencies defined in the description file to determine which files can be updated in parallel. For example, if the file *foo* is a dependency of the file *bar*, **pmake** will ensure that *foo* and *bar* are not simultaneously updated. For this reason, it is important that the description file dependencies be complete when using the **-j** and **-P** switches.

When using the **-P** switch, **pmake** becomes a parallel application. If you use the **-P** switch after this on a recursive invocation of **pmake**, it is ignored since **pmake** is already executing as a parallel application. Therefore, when invoking **pmake**, choose the best recursion level at which to use the **-P** switch. For example, if **pmake** is invoked with "**-j2 -P**.compute" switches on the following description file, the two targets *big* and *little* will be updated simultaneously.

```
all: big little

big:
      cd bigdir; $(MAKE)
little:
      cd littledir; $(MAKE)
```

If, in the above example, the *little* target is quickly updated, while the *big* target involves lots of compiles and takes several hours to build, the benefits of parallelism will be lost.

A better approach in this case is to invoke the top-level **pmake** without the **-j** or-**P** switches and use the switches at the second level instead. The better approach is shown below.

```
all: big little

big:
      cd bigdir; $(MAKE) -j8 -P.compute

little:
      cd littledir; $(MAKE) -j2 -P.compute
```

Commands

The commands to remake a target may be prefaced by one or more of the following special characters. The special characters are not passed to the shell but have the following effect within **pmake**:

-	**pmake** ignores any non-zero error code returned by the command.
+	**pmake** executes the command even if the **-n**, **-q** or **-t** switches are specified.
@	**pmake** does not print the command before executing it.

Included Description Files

Description files may be included within other description files by using the **include** directive as shown below. When **pmake** encounters an **include** directive within a description file, it temporarily stops processing the first description file, processes the included description file, and then resumes processing the original description file.

include *filename*	A file named *filename* is included and processed. An error occurs if the file is not found.
-include *filename*	A file named *filename* is included and processed. No error occurs if the file is not found.

Configuration File Support

When you invoke **pmake**, it searches for the configuration file *Makeconf*. It searches the build tree starting from the current directory and continuing up to the root directory. It includes the first *Makeconf* file it finds and processes this file prior to processing any description file. Not having a *Makeconf* file does not produce an error. This file contains rules that override the default rules **pmake** uses.

The **pmake** command allows a software project to be organized into separate source and object directory trees. The source tree contains files that are read but not modified by **pmake**. The object tree contains the target files that **pmake** creates or updates. The trees are assumed to have the same structure so that each source directory has a counterpart with the same name within the object tree. If the *Makeconf* file contains a definition for the variable *OBJECTDIR*, it is interpreted as the root of the object directory tree. *OBJECTDIR* may be defined either as an absolute path or as a path relative to the location of the *Makeconf* file.

Before running any commands, **pmake** computes the relative path from the location of the *Makeconf* file to the current directory and appends that path to the root of the object directory tree to determine the current object directory. The **pmake** command then changes directories to the object directory, creating subdirectories as needed, and modifies the *123* to search for source files in the original directory. For example, assume a *Makeconf* file in */usr/foo* defines *OBJECTDIR* as */usr/obj*. If you invoke **pmake** from */usr/foo/bar*, the object directory is */usr/obj/bar* and */usr/foo/bar* will be added to the *VPATH*.

The *Makeconf* file may include a definition for the variable *SOURCEDIR*. The *SOURCEDIR* variable is interpreted as a colon-separated list of alternate source directory trees. As with *OBJECTDIR*, *SOURCEDIR* may include both absolute pathnames and pathnames relative to the location of the *Makeconf* file.

If *SOURCEDIR* is defined, **pmake** computes the additional source directories to be searched in much the same manner as it computes the current object directory. The relative path from the location of the *Makeconf* file to the current directory is computed and then appended to each of the colon-separated path names in the *SOURCEDIR* variable. The resulting pathnames are then added to the *VPATH* and searched after the current directory.

Macros

Macros may be used to simplify and improve the portability of description files. Macros may be defined either on the command line or from within the description file. Macro definitions can have the following general forms:

PMAKE *(cont.)* PMAKE *(cont.)*

MACRO = *value*

Recursively expanding macro definition. The macro value is installed verbatim; if it contains references to other macros, those references are expanded when the macro is evaluated.

MACRO := *value*

Simply expanding macro definition. The value is evaluated once, when the macro is installed; imbedded macro references are evaluated at that time.

override *MACRO* = *value*

Override directive. Causes macro definition within a description file to override definition from the command line.

Macro references take the form $(*macro-name*) or ${*macro-name*} and may appear anywhere within the description file. Again, several special forms are supported.

$(...$(*MACRO*)...)

Nested macro references.

$(*MACRO*:*suffix1*=*suffix2*)

Suffix replacement references. The value of *suffix1* is replaced by *suffix2* in the expansion of *MACRO*. The *suffix1* must occur at the end of a word.

$(*MACRO*:*pattern1*=*pattern2*)

Pattern replacement references.The *pattern1* and *pattern2* values each contain the wild card character, %. Occurrences of *pattern1* in the expansion of *MACRO* are replaced by *pattern2* with the % character matching any stem.

$(*MACRO*/*reg-expression*/*replacement*)

Pattern replacement references. The replacement string is substituted for the reg-expression within the macro expansion. The valid forms of regular expressions are described in **regexp(3)**. Semicolons may be used in place of the slashes that separate the *MACRO*, *reg-expression*, and *replacement* strings.

$(*MACRO*:*X*) C-shell style modifiers. *X* may be **t** (tail), **h** (head), **r** (root) or **e** (extension).

$(*MACRO*?*value1*:*value2*)

Conditional expressions. Evaluates to *value1* if *MACRO* is defined and *value2* otherwise.

Special Macros

The following internal macros are automatically set as each target is processed:

$@	The name of the current target.
$*	The base name of the current target.
$<	The name of the current dependency file.
$?	The list of dependencies that are newer than the target.
$%	The name of the library member being processed.
$^	The list of all dependencies.
$$@	The current target (valid only on the dependency line).

Special Variables

Some variables have special meaning for the **pmake** command. Some of these variables are set automatically by the **pmake** command or they can be set as environment variables. The following special variables are supported:

cputype	The CPU type of the target system in lower-case (for example, `i860`). This variable is set automatically by the **pmake** command.
CPUTYPE	The CPU type of the target system in upper-case (for example, `I860`). This variable is set automatically by the **pmake** command.
MAKE	The command line with which **pmake** was invoked, excluding switches. This variable is set automatically by the **pmake** command.
MAKEFILES	A list of description files to be read before any others. This variable may be set in the environment.
MAKEFLAGS	A list of the switches specified on the command line. This variable is set automatically by the **pmake** command.
MAKELEVEL	The current level of make recursion. This variable is set automatically by the **pmake** command.
OBJECTDIR	The root of the object tree where **pmake** will build its targets.

SHELL	The shell to use for command execution. The default is */bin/sh*. This variable may be set in the environment or in a description file.
SOURCEDIR	The roots of the source tree where **pmake** will search for sources.
SUFFIXES	The list of default, known suffixes from built-in suffix rules. This variable is set automatically by the **pmake** command.
target_machine	The machine architecture of the target system in lower-case (for example, `paragon`). This variable is set automatically by the **pmake** command.
TARGET_MACHINE	The machine architecture of the target system in upper-case (for example, `PARAGON`). This variable is set automatically by the **pmake** command.
VPATH	A colon-separated list of directories to search for dependency files. This variable may be set in the environment or in a description file.

Pseudotarget Names

The **pmake** command assigns special meanings to the following pseudotargets:

.DEFAULT	If it appears in the description file, the rule for this target is used to process a target when there is no other entry for it.
.EXIT	If defined in the description file, **pmake** processes this target and its dependencies after all other targets are built.
.EXPORT	Variables listed as dependencies of this target are expanded and exported to the environment in which **pmake** runs it's commands. Unlike GNU **make**, **pmake** does not normally export variables defined within a definition file.
.IGNORE	Ignore errors. When this target appears in the description file, **pmake** ignores non-zero error codes returned from commands.
.INIT	If defined in the description file, this target and its dependencies are built before any other targets are processed.
.PHONY	The dependencies of this target are considered to be "phony" targets. When it is time to consider such a target, **pmake** will run its commands unconditionally regardless of whether a file with that name exists or what its last modification time is.

.PRECIOUS List of files not to delete. **pmake** does not remove any of the files listed as dependencies for this target when interrupted. **pmake** normally removes the current target when it receives an interrupt.

.SILENT Run silently. When this target appears in the description file, **pmake** does not echo commands before executing them.

.SUFFIXES The dependencies of this target are the suffixes that **pmake** will search for when applying suffix rules.

Conditional Execution

The **pmake** utility provides conditional execution directives that control what parts of the description file **pmake** sees. The general format of a conditional directive is the following:

conditional-directive
text-if-true
endif

Another format is the following:

conditional-directive
text-if-true
else
text-if-false
endif

There are four different conditional directives. They are:

ifeq (*arg1*, *arg2*)
The conditional evaluates to true if *arg1* is equal to *arg2*.

ifneq (*arg1*, *arg2*)
The conditional evaluates to true if *arg1* is not equal to *arg2*.

ifdef *variable-name*
The conditional evaluates to true if *variable-name* is defined.

ifndef *variable-name*
The conditional evaluates to true if *variable-name* is not defined.

Archive Support

Archive library members may be referenced within a description file using the following form:

`lib(member)`

Functions

Functions provide support for text processing within a description file. The **pmake** utility provides approximately 20 functions including functions to do pattern replacement, filtering, sorting and filename component selection. The general form of a function call is: $(*function arguments*) or ${*function arguments*}.

For a description of the available functions and their arguments, refer to *The GNU Make Manual.*

See Also

The GNU Make Manual, available through the Intel Supercomputer Systems Division Customer Service Response Center (*support@ssd.intel.com*).

Copyright

The **pmake** utility is an extension of GNU **make** and is distributed under the terms of the GNU General Public License. Intel will provide a complete copy of the **pmake** source code upon request. For more information, contact Intel's SSD Customer Service Response Center.

PSPART PSPART

Shows status of the applications in a partition.

Syntax

pspart [*partition*]

Arguments

partition Absolute or relative pathname of a partition. If *partition* is not specified, *NX_DFLT_PART* is used. If *NX_DFLT_PART* is not defined, *.compute* is used.

Description

The **pspart** command shows the following status information about all of the applications running in a partition:

`PGID` The process group ID of the application. The process group ID of an application is always the same as the process ID of the application's controlling process.

`USER` The login name of the user who invoked the application.

`SIZE` The number of nodes allocated to the application from the partition.

`PRI` The application's priority.

`TIME ACTIVE` The amount of time the application has been active (rolled in) in the current rollin quantum. The time active is shown both in the format [[*hours* :]*minutes* :]*seconds* . *milliseconds* and as a percentage of the partition's rollin quantum. If the application is not active in the current rollin quantum, a dash (-) is shown for both quantities.

`TOTAL TIME` The total amount of time the application has been rolled in since it was started, in the format [[*hours* :]*minutes* :]*seconds* . *milliseconds*.

`COMMAND` The command line by which the application was invoked.

You must have read permission on the specified partition to use this command.

Example

The following command shows the status of the applications in the partition *mypart*, whose parent partition is the compute partition:

```
pspart mypart

   PGID USER        SIZE PRI     TIME ACTIVE    TOTAL TIME  COMMAND
  12345 smith        256   5      45.00  75%       4:41.60  /home/smith/glide
  23456 joandoe       67   4          -    -       7:12.30  boggle -sz 67
  34567 noname       192  10   1:00.00 100%     2:12:3.90  myfft -sz 192
```

In the example above, the partition *mypart* has a rollin quantum of one minute. The application */home/smith/glide* has been active for 45 seconds, or 75% of the rollin quantum; the application *boggle* is not currently active; and the application *myfft* has been active for one minute, or 100% of the rollin quantum.

Errors

`pspart: Partition not found`

You specified a partition that does not exist.

`pspart: Partition permission denied`

You do not have read permission for that partition.

See Also

chpart, lspart, mkpart, ps(1), rmpart, showpart

RESET RESET

Resets the Intel supercomputer from the diagnostic station.

Syntax

reset [ramdisk | flash | skip]

Arguments

ramdisk Resets the Intel supercomputer and boots the system ramdisk. Use this argument for installing a ram-resident operating system.

flash Resets the Intel supercomputer but does not boot it. This executes the **async** command and attaches your console to the serial line. Use this argument to maintain flash programs.

skip The Intel supercomputer is reset, but the final connection to the console is not made. Use this argument for applications that have their own interface to the console.

Description

The **reset** command is for use by the system administrator on the diagnostic station only.

The **reset** command resets an Intel supercomputer. Before running the **reset** command, edit special variables in the **reset** script file to reflect the system configuration.

The **reset** script begins with three sections. The first section contains variables you may modify to set the system configuration. The second section contains variables you may change, but rarely need changing. The third section contains variables you should never modify. Comments above each section repeat this description.

In the first section of the **reset** script, set the following variables:

`CONFIGURATION={full|multi|condo}`
Specifies the system configuration. For a full system (four backplanes), set this variable to `full`. If your system has two or three backplanes, set this variable to `multi`. If your system has one backplane, set this variable to `condo`.

RESET *(cont.)* RESET *(cont.)*

`TOP_BACKPLANE={A|B|C|D}`
Specifies the backplane number for the top backplane. For full configurations, set this variable to `D`.

`BOOT_DIR=/usr/paragon/boot`
Specifies the boot directory pathname. This may not need to be changed, but sometimes disk space is at a premium and files are stored elsewhere. Set this variable to the directory where the kernel, ramdisk and reset scripts reside.

`SERIAL_DEVICE=/dev/tty1a`
Specifies the name (absolute pathname) of the device where the serial line from the Intel supercomputer is attached.

`HAS_RPM={1|0}`
Specifies whether the Intel supercomputer has RPM hardware. If your system has RPM hardware, set this variable to `1`. Otherwise, set this variable to `0`.

Remove the **echo** command and the **exit** command in the **reset** script when you are done editing the script. These commands are used to remind you to edit the **reset** script.

Files

/usr/paragon/boot/reset	The script file for the **reset** command.

See Also

async, **bootpp**, **getmagic**

RMPART RMPART

Removes the named partition.

Syntax

rmpart [**-f**] [**-r**] *partition*

Arguments

-f	Removes a partition and any applications running in the partition. When you use the **-f** switch, the **rmpart** command terminates all the applications running in the specified partition and then removes the named partition. By default, the **rmpart** command does not remove partitions in which applications are running.
-r	Removes a partition and all its subpartitions if no applications are running in the partition or its subpartitions. This is a recursive operation; all subpartitions and their subpartitions in the specified partition are removed. When used with the **-f** switch, the **rmpart** command terminates all the applications running in the partition and its subpartitions, and removes the partition and all its subpartitions.
partition	The absolute or relative partition pathname of the partition to be removed.

Description

This command removes the named partition. Use the **-f** switch if the partition contains running applications. Use the **-r** switch to remove a partition and all its subpartitions.

To remove a partition, you must have write permission on its parent partition.

Examples

If there are applications running in *mypart*, use the following command to terminate the applications and remove the partition:

```
rmpart -f mypart
```

Use the following command to remove the *mypart partition* and all its partitions:

```
rmpart -r mypart
```

Errors

`rmpart: Partition lock denied`

You specified a partition that is currently in use and being updated by someone else. You cannot remove a partition that is currently being updated.

`rmpart: Partition not empty`

You specified a partition that contains subpartitions or has active processes.

`rmpart: Partition not found`

You specified a partition that does not exist.

`rmpart: Partition permission denied`

You specified a partition whose parent partition does not grant you write permission.

See Also

chpart, lspart, mkpart, pspart, showpart

SAT SAT

Runs the Paragon XP/S system acceptance test.

Syntax

sat [**-bchxV**] [**-d** *dir*] [**-l** *log*] [**-m** *mins*] [**-o** *output*] [**-p** *partition*] [**-r** *reps*] [*test* . . .]

Arguments

-b	Causes the **sat** command to build tests from sources prior to running them.
-c	Runs tests concurrently. You must have root permissions to use the **-b** switch.
-h	Displays a help message and exits.
-x	Causes the acceptance test to terminate if a test fails. By default, testing continues after an error is reported in the transcript.
-V	Causes the **sat** command to write version information to standard output and then exit.
-d *dir*	Specifies a user-defined directory from which to run acceptance tests. Otherwise, the **sat** executes tests from the directory */usr/lib/sat*.
-l *log*	Causes the **sat** command to write a transcript of the test session to a *log* file.
-m *mins*	Specifies the length of time, in minutes, to run the tests. By default, the tests execute once without regard for time.
-o *output*	Causes the **sat** command to write a final report to a user-specified output file.
-p *partition*	Specifies the partition in which the **sat** parallel tests are run. Otherwise, parallel tests run in the partition specified by the environment variable *NX_DFLT_PART*. If *NX_DFLT_PART* is not defined, tests run in the .compute partition.
-r *reps*	Sets the number of times to repeat the tests. By default, tests execute once.
test . . .	Specifies one or more particular system acceptance tests to run.

Exit Values

An exit value of

- 0 indicates successful completion of all tests.
- 1 indicates test failures.
- 2 indicates an error occurred while processing command arguments or scanning the directory hierarchy.

Files

/usr/lib/sat/README contains a description of the Paragon XP/S SAT

/usr/lib/sat/help contains help information

Directories

/usr/lib/sat root of default test directory hierarchy

/usr/tmp directory for temporary files created by the **sat** command

General Description

The Paragon XP/S system acceptance test (SAT) provides a set of test suites designed to rigorously test Paragon XP/S system functionality and performance. The standard tests, which are run on the system before it leaves the factory, can be used to test the system after it is installed at your site, after installing additional hardware or software, and to perform periodic system performance tests. The SAT is invoked with the **sat** command. The **sat** command is a test driver, located in */usr/bin/sat* that executes tests in the directory */usr/lib/sat*, or in a directory you specify with the **-d** switch.

The **SAT** test hierarchy consists of organizing directories and test directories. Organizing directories are used to organize the test directories. Test directories contain run scripts, executable files, data files, sources, and makefiles for compiling and running tests. The sat command identifies a test directory by the presence of a *run* file and a *README* file.

The **sat** command creates a variety of temporary files while running. These files, along with the temporary files created by the tests in */usr/lib/sat*, are created in */usr/tmp*. When it exits, the **sat** command removes any temporary files it created.

Using the sat Command Switches

Following are descriptions of the **sat** command switches:

Running Specific Tests (*test*)

If you don't specify a test, the SAT runs all tests. If you specify one test, **sat** runs only that test. For example, to run only the Level 2 BLAS test, you would enter:

```
sat blas2
```

To run several specific tests, enter a space between the names of the tests you want to run; for example:

```
sat blas2 paranoia
```

Displaying Help Information (-h)

The **-h** switch displays a help message. If you select **-h** and specify a test, **sat** writes a help message to standard output that briefly describes the test. If no tests are specified, the output message describes the **sat** command, usage information, and a list of tests available in the directory hierarchy. The **sat** command exits after displaying these help messages.

Building Tests from Sources (-b)

The **-b** switch causes the **sat** command to build tests from sources prior to running them. Root permissions are required to build tests. Only the tests that include a *makefile* (or *Makefile*) along with the sources are built. If you specify **-b** with **-r** (repetitive test runs), the build occurs once, before running the tests. If you specify **-r** 0, the SAT will build the tests without running any of the tests.The build time does not affect the number of times the tests are run. If you specify **-m** (run tests for length of time in *minutes*) with **-b**, the build occurs once, before running the tests. The build time does not affect the time allotted to run the tests.

SAT *(cont.)* **SAT** *(cont.)*

NOTE

As shipped from the factory, the SAT includes the necessary sources and executables for each test. Although the SAT is designed to remove executables prior to each build, the makefiles installed at the factory make sure the required compilers are present before removing any binaries. This prevents the SAT from accidentally destroying the executables on systems that do not have the required compilers.

Running Tests Concurrently (-c)

By default, the **sat** command runs tests in sequential mode. The **-c** switch sets concurrent operation of tests, which means all the tests start simultaneously. Concurrent operation implies that the tests will compete for system resources. Thus, if you run the tests concurrently (or if the system is running other applications when you run the tests) system performance is likely to be affected. The SAT still reports accuracy and completeness results.

Repeating Tests (-m, -r)

The **-m** switch specifies the length of time, in minutes, to repeat the tests. The time parameter must be specified greater than or equal to 1 minute. When the specified time limit is reached, the **sat** command terminates the test even if multiple tests or concurrent operation have been specified.

The **-r** switch sets the number of times to repeat the test(s). The repetitions must be set greater than or equal to 0. Specifying -r 0 in conjunction with -b (build tests), allows you to build tests without actually running the SAT. Tests repeat to the specified count whether they pass or fail. If you specify -r with concurrent operation (-c), the tests will run in concurrent mode until all tests have reached the specified count.

If you specify both **-r** and **-m**, execution continues until either the repetition or time condition is met.

If you specify **-b** (build tests) with **-m** or **-r**, the time to build the tests does not impact the time or count specified.

If **-x** (exit on error) is specified with **-m** or **-r**, and an error is encountered, the entire run exits regardless of the time or count specified.

SAT *(cont.)* SAT *(cont.)*

Specifying a Partition for Parallel Tests (-p)

Parallel **sat** tests are run in a partition. The **-p** switch identifies the test partition. Otherwise, parallel tests run in the partition specified by the environment variable *NX_DFLT_PART*. If that variable is not defined, the tests run in the *.compute* partition. You must have permission to execute programs in the specified partition. Non-parallel tests are executed in the *.service* partition. Refer to the *Paragon™ OSF/1 User's Guide* for more information about the *.compute* and *.service* partitions.

Running Tests from a User-Specified Directory (-d)

By default, the **sat** executes tests from the directory hierarchy starting at */usr/lib/sat*. The **-d** switch allows you to specify a different directory from which to run acceptance tests. If you specify a non-existent directory, the **sat** will exit and generate an error message.

Creating Log Files and Final Reports (-l, -o)

The **-l** and **-o** switches make it possible to write a transcript of a SAT test session to a file. The file produces by the **-l** switch is called a *log file*. The log file contains a copy of the transcript of the test session. For example, if you wanted to run the Livermore LOOPS test and save the transcript to a file called *logfile*, you would enter:

```
sat -l logfile testname
```

The -o switch produces an output file, called a final report, that contains expanded information about the test that was run. If you specify, **-o** *outfile*, **the sat command sends** complete test results, operating parameters, and configuration information to *outfile*. This output file is called a *final report*. For example, if you wanted to obtain a more complete set of results from the Livermore LOOPS test, you could enter:

```
sat -o yourfile lloops
```

Exiting on Error (-x)

The **-x** switch causes **sat** to terminate and exit as soon as any errors are found. If **-x** is not used, testing continues after the error is reported.

Displaying Version Information (-V)

Specifying the **-V** switch displays the version of the SAT your system is using.

SAT *(cont.)*

SAT *(cont.)*

See Also

Paragon™ XP/S System Acceptance Test User's Guide

SCANIO SCANIO

Establishes a scan-based console interface between the diagnostic station and a node on an Intel supercomputer.

Synopsis

scanio [**-eiqv**] [**-b** *file*] [**-c** [*letter*]] [**-f** *file*] [**-s** *string*] [**-t** *state*] [*node*]

Arguments

-b *file* — Specifies the *bootmagic* file pathname. The default is the bootmagic file in the current directory.

-c [*letter*] — Specifies the backplane to use for a condo configuration. The letter argument can be `A`, `B`, `C`, or `D`. Backplane A is closest to the floor. By default, the **scanio** command assumes the node numbers begin in the upper left corner of the last cabinet (the one furthest away from the diagnostic station). (Some sites have multiple Intel supercomputer systems within the same cabinet. Although this is not a supported configuration, the **scanio** command provides minimal support for this configuration.)

-e — Specifies that the **scanio** command exit after the node has been silent for a few seconds. Keyboard input is not accepted, however, an INTR interrupt from the keyboard kills the **scanio** command.

-f *file* — Specifies reading characters from the file specified by the *file* argument rather than from the keyboard. It's normally used to define boot variables during system booting.

-i — Specifies that the *BOOT_CONSOLE* variable is not checked to make sure the kernel is setup for using the scan interface. The **reset** command uses this feature.

-q — Suppresses messages. This is the opposite of the **-v** switch.

-s *string* — Specifies using the *string* argument to set boot variables. This switch is similar to the **-f** switch except that the data contained in the *string* argument is used in place of keyboard input.

-t *state* — Tests the state of the INT_FROM_NODE line. The *state* argument can be either `1` or `0` (zero). When the *state* argument is set to 0, the **scanio** command returns 0 if the INT_FROM_NODE is high and `1` if it is low. When the *state* argument is set to `1`, the **scanio** command returns `1` if the INT_FROM_NODE is high, and 0 if it is low.

SCANIO *(cont.)* SCANIO *(cont.)*

-v Turn verbosity on.

node Node number of the node the console is connected to.

Description

The **scanio** command is for use by the system administrator on the diagnostic station only.

The **scanio** command establishes serial communications with any node in the Intel supercomputer system using the scan lines rather than the mesh.

When you invoke the **scanio** command, the command examines the bootmagic file to determine the size and configuration of the Intel supercomputer system. The following variables are extracted from the bootmagic file:

BOOT_CONSOLE
This must be set to `s`. This tells the kernel that the console interface will use the scan lines.

BOOT_NODE_LIST
This variable determines which nodes the system uses. When you attempt to connect to a node, the node number you enter is validated against this list.

MESH_X The number of cabinets calculated as `MESH_X/4`

MESH_Y The number of backplanes is calculated as `MESH_Y/4`.

BOOT_FIRST_NODE
This variable determines the default node to connect to if the user does not specify a node.

NOTE

The speed of the **scanio** command is somewhere between 1200 and 2400 baud. Therefore, **scanio** is very slow.

SCANIO *(cont.)*

SCANIO *(cont.)*

Node Numbering

Node numbers can be specified in CBS (Cabinet, Backplane, Slot) format (see the **cbs** manual page), or root node numbers. Root node numbers are the numbers used in the bootmagic file.

Regardless of which node number format you use, node numbers are validated against the list of node numbers extracted from the bootmagic file. You are not allowed to connect to any node other than the nodes defined by *BOOT_NODE_LIST*.

Internal Commands

The **scanio** command sends everything you type to the current node, and displays all text from the node to your standard output. You can enter special commands by typing a tilde (~) immediately after hitting the return key, then typing a special command character. The following command characters are currently supported:

~!
: Allows you to invoke a system command. After the system command completes, control returns to the node. You can invoke a shell if you want to execute more than one command.

~#
: Allows you to switch nodes. The **scanio** command displays a list of valid nodes along with a prompt. You can use root node numbers or CBS numbers. This list of valid nodes is displayed in root node numbering format.

Bugs And Problems

The **scanio** command runs between 1200 and 2400 baud. This happens because every read across the scan line requires a write. For more information, refer to IEEE 1149.1 1990 Standard.

The **scanio** command never sleeps. It spins between looking at the scan lines and looking at the input device (usually the keyboard). This happens because the node being used cannot interrupt the operating system on the diagnostic station when data is ready.

See Also

async, cbs

SHOWPART SHOWPART

Shows a partition's characteristics.

Syntax

showpart [*partition*]

Arguments

partition Absolute or relative pathname of a partition. If you do not specify a partition argument, the command uses the value specified in the *NX_DFLT_PART* environment variable. If *NX_DFLT_PART* is not set, the default is *.compute*.

Description

The **showpart** command shows information about the characteristics of a specified partition. A partition's characteristics include the name of the partition's owner, the owner's group, the partition's access rights, the number of nodes in the partition, the partition's rollin quantum value, and the effective priority limit.

The command displays a picture that shows the partition's size, shape, and position in the system. the picture has the following format:

- A large rectangle represents the root partition.
- Numbers to the left of the rectangular picture show the root partition node numbers of the nodes in the first column of each row.
- Asterisks (*) within the rectangle represent nodes that are allocated to the specified subpartition.
- Dashes (-) represent empty slots.
- Xs represent bad or unusable nodes that are allocated to a partition.
- Dots (.) represent other nodes.

You must have read permission on the specified partition to use this command.

SHOWPART *(cont.)* SHOWPART *(cont.)*

Examples

To show the characteristics of the partition called *mypart*, whose parent partition is the compute partition, you can use the following command:

```
showpart mypart

USER    GROUP   ACCESS  SIZE    RQ      EPL
smith   eng     777     9       15m     5

     +---------+
    0| . . . . |
    4| . * * * |
    8| . * * * |
   12| . * * * |
     +---------+
```

In this case, *mypart* belongs to user *smith* in group *eng*. It has permissions 777 (`rwxrwxrwx`), a size of 9 nodes, a rollin quantum of 15 minutes, and an effective priority limit of 5.

The rectangular picture at the bottom of the **showpart** output shows the size, shape, and position of the *mypart* subpartition within the root partition:

- The root partition is 4 nodes high and 4 nodes wide.
- The *mypart* partition consists of nodes 5–7, 9–11, and 13–15 of the root partition.

The following command has the same output as the first example using an absolute partition pathname:

```
showpart .compute.mypart
```

SHOWPART *(cont.)* SHOWPART *(cont.)*

Errors

`showpart: Partition not found`

You specified a partition that does not exist.

`showpart: Partition permission denied`

You do not have read permission for the specified partition.

See Also

chpart, lspart, mkpart, pspart, rmpart

Index

A

application 1

async 8

B

bootmesh 10

bootpp 12

C

cbs 17

chpart 19

F

fsplit 22

I

ipd 25

L

lsize 26

lspart 28

M

mkpart 31

P

parsemagic 35

pmake 37

pspart 50

R

reset 52

rmpart 54

S

sat 56

scanio 62

showpart 65

WE WOULD LIKE YOUR COMMENTS

We are trying to produce the best documentation to meet your needs. Please take a few moments to help us out by providing the information requested below. Our Internet address is **techpubs@ssd.intel.com**

Manual Title: _____

Your Comments

Please describe any information that you feel should be added to this document (please indicate where the information can be found): _____

_____

_____

Please describe any areas of this document that you feel need improvement (please specify chapter number/name, page number, and location on page): _____

_____

_____

Please describe any errors you found (please specify chapter number/name, page number, and location on page): _____

_____

_____

Other comments: _____

_____

Information About You

What is your job title? _____

How did you use this document? _____

If you would like a response from us, please provide the following information:

Name _____

Title _____

Company _____ Department/Mail Station _____

Address _____

City _____ State _____ ZIP Code _____

Country _____ Phone ( ) _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 42 BEAVERTON, OR

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Supercomputer Systems Division
Technical Publications, MS: CO1-01
15201 N.W. Greenbrier Parkway
Beaverton, OR 97006

Please fold here and close the card with tape. Do not staple.

If you are in the United States and are sending only this card, postage is prepaid.

If you are sending additional material or if you are outside the United States, please place this card and any additional material in an envelope. Send the envelope to the address printed on this form, adding "United States of America" if you are outside the United States.