1. The int**e**nt® Shell Commands

The int**e**nt® shell is a scripting command language interpreter. It is able to read and execute commands from the user, and can therefore provides an interface to the underlying mechanisms. This document contains a full listing of all such available, commands and has been designed to be read in conjunction with *"The* int**e**nt™ *Shell User Guide*".

In addition to the shell commands described here, there are certain utilities which can be run from the shell but which are not part of the shell. These include:

- The Kernel Entropy Collector
- The Data Flow Analysis Utility
- The Test Coverage Analysis Utility
- The VP Assembler
- The Disassembler
- Editors - ed and jove

2. Conventions

Shell commands can be entered at the $ prompt, as displayed on the screen within the int**e**nt® box. Where a command line is shown, text enclosed in angle brackets *<thus>* should be replaced by an actual parameter when typing the command. Parts of the command line shown in square brackets *[thus]* are optional. "..." indicates optional repetition of the previous item. "|" separates alternatives. Braces *{thus}* may be used to group items.

2.1.1 Example

```
command <parameter1> [<parameter2> [<parameter3> ...]]
```

In this case parameter 1 is mandatory, and parameters 2 and 3 are optional. However, parameter 3 cannot be specified unless parameter 2 also is. Parameter 3 may be repeated.

2.1.2 Options

Commands can be modified by specifying additional options. All options must be preceded by "-". Multiple options can be specified together, so for example "-abc" would behave identically to "-a -b -c". However some options may take additional arguments, in which case this may be impossible in the case of that particular option.

Options can appear anywhere on the command line, between parameters. If an option of "-" is found, no further parameters will be processed as options; this is essential if any parameter might actually start with a "-". For clarity, it is recommended that options be placed immediately after the command name, before any parameters.

3. Commands A and B

3.1 Analysemap - Front end to counttools to produce a report on tool sizes.

Synopsis

```
analysemap <mapfile>
```

Description

This script uses counttools to produce a breakdown of the code size for each part of the system. It searches the map file specified by <*mapfile*> to find all of the directories of tools that are listed. The size of all tools in these directories (and all subdirectories) are then listed. If not extension is specified, then '`.map`' will be added automatically.

Note that this script can take a long time to run, particularly for large images. Also note that to be useful, the output probably needs to be redirected.

Examples

The following command will analyse the map file *sys/platform/win32/pjavafull.map.*

```
analysemap sys/platform/win32/pjavafull > tools.txt
```

3.2 *and* - Execute Multiple Commands in Sequential Order

Synopsis

```
and <command> ...
```

Description

Executes the first specified command. If that exits with a zero status (indicating success), the second command (if any) is then executed. It will then continues until either all commands have been executed, or one of them has exited with a non-zero status. The status is the exit status of the last command that has been executed.

UNIX equivalent	`&&` for instance `{...cmd1...}&&{...cmd2...}`
DOS equivalent	None applicable

Example

```
and devinfo dev/_loadisr ps
```

This will first display information concerning the *_loadisr* device driver and then display information about the status of currently executing processes.

3.3 *ascii* - Display Table of ASCII Characters

Synopsis

```
ascii [ <value> | <min-value> <max-value> ]
```

Description

Displays a table of ASCII characters on standard output. If a single character value, or a range, is specified, then only the selected characters are displayed. By default, the range 32 to 126 inclusive (printable ASCII) is used. Non-printable characters are omitted in any case.

UNIX equivalent	case

DOS equivalent	None applicable

Options

-1	Display in a single column (i.e., one entry per line),rather than in multiple columns.
-d	Display character numbers in decimal instead of hexadecimal.

3.4 Atomdump - Displays the Elate® atom list.

Synopsis

```
atomdump [ < options > ]
```

Description

Examines the atom list and the static atom table, displaying for each atom the name and (optionally) atom value and (if applicable) the reference count and flags.

Options

-b	Display only the names of the atoms, and not the atom value, etc.

3.5 *basename* - Strip Directory and Suffix From Filenames

Synopsis

```
basename <pathname> [<suffix>]
```

Description

This function outputs, terminated by a newline, the last component of the specified pathname. If a suffix is specified and is present at the end of this component, it will be removed.

Example

```
basename x/y/z.c
```

Outputs to stdout *z.c*, prints just the filename out, and removes the directory part of the filename.

UNIX equivalent	`basename`
DOS equivalent	None applicable

3.6 *bind* - Bind Specified Tool

Synopsis

```
bind <tool-name> ...
```

Description

This function attempts to bind each specified tool in turn. By default, each tool's reference count is incremented, so that they cannot be unbound later.

UNIX equivalent	None applicable
DOS equivalent	None applicable

Options

-c	Do not increase the tool's reference count. This option would be used if one merely wishes to check whether it is possible to bind a tool.
-v	Echo to standard error the name of each tool successfully bound.

3.7 *break* - Break From Inner-Most Loop.

Synopsis

```
break
```

Description

Causes execution to move to the first statement after the body of the inner-most loop. The exit status of the loop is zero. This is done by throwing the exception *shell.flow.break*, and thus the *break* command does not have to be in the same function or file as the inner-most loop. The loop may be either a *while* loop or a *for* loop.

4. Commands C

4.1 *case* - Match String Against Patterns

Synopsis

```
case <string> [ { <pattern> <command> } ... ]
```

Description

This function matches the specified string against each of the patterns in turn. If one matches, its corresponding command will be executed. The exit status is the status of the command executed, (if any) or zero if the string didn't match any of the patterns.

UNIX equivalent	None Applicable
DOS equivalent	None applicable

4.2 *cat* - Concatenate Named Files

Synopsis

```
cat [<filename> ...]
```

Description

This function concatenates all the named files, or reads standard input if none have been specified. The resulting input will then be copied to standard output.

Example

```
cat app/stdio
```

UNIX equivalent	`cat`
DOS equivalent	`type`

Options

-A	Combines the options provided by -v,-E and -T.
-E	Output a "$" at the end of each line.
-T	Display tab characters as "^I".
-b	Number non-blank lines.
-e	Combines the options provided by -v and -E.
-n	Number all lines.
-s	Squeeze consecutive blank lines into a single blank line.
-t	Equivalent to -vT.
-u	Disable buffering of output.
-v	Display unprintable characters in a printable manner.

4.3 *catch* - Catch Exception

Synopsis

```
catch <try-command> <exception-variable> <status-variable> [ {<pattern>
<command> ... ] [<default-command>]
```

Description

Executes the <*try-command*>, and saves its exit status in the variable <*status-variable*>. If it exited normally, set <*exception-variable*> to an empty array, and return that status. If it exited with an exception, save the exception in <*exception-variable*>, and proceed thus:

Compare the exception string against each <*pattern*> in turn. If one matches, execute the corresponding <*command*>. If no pattern matches, execute the <*default-command*> if specified, or rethrow the caught exception if not.

It is only in rare cases that one would wish to override the default <*default-command*>. Normally, whatever processing the <*default-command*> does, it should finish by rethrowing the caught exception.

UNIX equivalent	None Applicable
DOS equivalent	None applicable

4.4 *cd* - Change Directory

Synopsis

```
cd [<directory>]
```

Description

This function changes the current directory to the specified directory. If no directory has been specified, the current directory is changed to the root directory instead.

The current directory is actually changed by setting the environment variable “sys.cwd.” As a side effect, the current directory can always be read from that variable.

Example

```
cd dev
```

UNIX equivalent	cd
DOS equivalent	cd

4.5 *chat* - Converses with a modem as directed by a script.

Synopsis

```
chat <script>
```

Description

The chat program defines a conversational exchange between the computer and the modem. Its primary purpose is to establish the connection between the Point-to-Point Protocol device and the remote ppp server.

Options

-f <chat file>	Read the chat script from the chat file. The use of this option is mutually exclusive with the chat script parameters. Multiple lines are permitted in the file. Space or horizontal tab characters should be used to separate the strings.
-t <timeout>	Set the timeout for the expected string to be received. If the string is not received within the time limit then the reply string is not sent. An alternate reply may be sent or the script will fail if there is no alternate reply string. A failed script will cause the chat program to terminate with a non-zero error code.
-r <report file>	Set the file for output of the report strings. If you use the keyword REPORT, the resulting strings are written to this file. If this option is not used and you still use REPORT

	keywords, the stderr file is used for the report strings.
-v	This option is ignored. (Request that the chat script be executed in a verbose mode. The chat program will then log all text received from the modem and the output strings which it sends to the SYSLOG.)
-V	Request that the chat script be executed in a stderr verbose mode. The chat program will then log all text received from the modem and the output strings which it sends to the stderr device.
script	If the script is not specified in a file with the -f option then the script is included as parameters to the chat program.

The chat script

The chat script defines the communications.

A script consists of one or more "expect-send" pairs of strings, separated by spaces, with an optional "subexpect- subsend" string pair, separated by a dash as in the following example:

```
ogin:-BREAK-ogin: ppp ssword: hello2u2
```

This line indicates that the chat program should expect the string "ogin:". If it fails to receive a login prompt within the time interval allotted, it is to send a break sequence to the remote and then expect the string "ogin:". If the first "ogin:" is received then the break sequence is not generated.

Once it received the login prompt the chat program will send the string ppp and then expect the prompt "ssword:". When it receives the prompt for the password , it will send the password hello2u2.

A carriage return is normally sent following the reply string. It is not expected in the "expect" string unless it is specifically requested by using the \r character sequence.

The expect sequence should contain only what is needed to identify the string. Since it is normally stored on a disk file, it should not contain variable information. It is generally not acceptable to look for time strings, network identification strings, or other variable pieces of data as an expect string.

To help correct for characters which may be corrupted during the initial sequence, look for the string "ogin:" rather than "login:". It is possible that the leading "l" character may be received in error and you may never find the string even though it was sent by the system. For this reason, scripts look for "ogin:" rather than "login:" and "ssword:" rather than "password:".

A very simple script might look like this:

```
ogin: ppp ssword: hello2u2
```

In other words, expect ....ogin:, send ppp, expect ...ssword:, send hello2u2.

In actual practice, simple scripts are rare. At the vary least, you should include sub-expect sequences should the original string not be received. For example, consider the following script:

```
ogin:--ogin: ppp ssword: hello2u2
```

This would be a better script than the simple one used earlier. This would look for the same login: prompt, however, if one was not received, a single return sequence is sent and then it
will look for login: again. Should line noise obscure the first login prompt then sending the empty line will usually generate a login prompt again.

Abort strings

Many modems will report the status of the call as a string. These strings may be CONNECTED or NO CARRIER or BUSY. It is often desirable to terminate the script should the modem fail to connect to the remote. The difficulty is that a script would not know exactly which modem string it may receive. On one attempt , it may receive BUSY while the next time it may receive NO CARRIER.

These "abort" strings may be specified in the script using the ABORT sequence. It is written in the script as in the following example:

```
ABORT BUSY ABORT 'NO CARRIER' '' ATZ OK ATDT5551212

CONNECT
```

This sequence will expect nothing and then send the string ATZ. The expected response to this is the string OK. When it receives OK, the string ATDT5551212 to dial the telephone. The expected string is CONNECT. If the string CONNECT is received the remainder of the script is executed. However, should the modem find a busy telephone, it will send the string BUSY. This will cause the string to match the abort character sequence. The script will then fail because it found a match to the abort string. If it received the string NO CARRIER, it will abort for the same reason. Either string may be received. Either string will terminate the chat script.

Report strings

A report string is similar to the ABORT string. The difference is that the strings, and all characters to the next control character such as a carriage return, are written to the report file.

The report strings may be used to isolate the transmission rate of the modem's connect string and return the value to the chat user. The analysis of the report string logic occurs in conjunction with the other string processing such as looking for the expect string. The use of the same string for a report and abort sequence is probably not very useful, however, it is possible.

The report strings do not change the completion code of the program.

These "report" strings may be specified in the script using the REPORT sequence. It is written in the script as in the following example:

```
REPORT CONNECT ABORT BUSY '' ATDT5551212 CONNECT ''

ogin: account
```

This sequence will expect nothing; and then send the string ATDT5551212 to dial the telephone. The expected string is CONNECT. If the string CONNECT is received the remainder of
the script is executed. In addition, the program will write to the expect-file the string "CONNECT" plus any characters which follow it such as the connection rate.

Timeout

The initial timeout value is 45 seconds. This may be changed using the -t parameter.

To change the timeout value for the next expect string, the following example may be used:

```
ATZ  OK ATDT5551212 CONNECT TIMEOUT 10 ogin:--ogin:

TIMEOUT 5 assword: hello2u2
```

This will change the timeout to 10 seconds when it expects the login: prompt. The timeout is then changed to 5 seconds when it looks for the password prompt.

The timeout, once changed, remains in effect until it is changed again.

Sending EOT

The special reply string of EOT indicates that the chat program should send an EOT character to the remote. This is normally the End-of-file character sequence. A return character is not sent following the EOT. The EOT sequence may be embedded into the send string using the sequence ^D.

Generating break

The special reply string of BREAK will cause a break condition to be sent. The break is a special signal on the transmitter. The normal processing on the receiver is to change the transmission rate. It may be used to cycle through the available transmission rates on the remote until you are able to receive a valid login prompt. The break sequence may be embedded into the send string using the \K sequence.

Escape sequences

The expect and reply strings may contain escape sequences. All of the sequences are legal in the reply string. Many are legal in the expect. Those which are not valid in the expect sequence are so indicated.

''	Expects or sends a null string. If you send a null string then it will still send the return character. This sequence may either be a pair of apostrophe or quote characters.
\b	represents a backspace character.
\c	Suppresses the newline at the end of the reply string. This is the only method to send a string without a trailing return character. It must be at the end of the send string. For example, the sequence hello\c will simply send the characters h, e, l, l, o. (not valid in expect.)
\d	Delay for one second. The program uses sleep(1) which will delay to a maximum of one second. (not valid in expect.)
\K	Insert a BREAK (not valid in expect.)
\n	Send a newline or linefeed character.
\N	Send a null character. The same sequence may be represented by \0. (not valid in expect.)
\p	Pause for a fraction of a second. The delay is 1/10th of a second. (not valid in expect.)
\q	Suppress writing the string to the SYSLOG file. The string ?????? is written to the log in its place. (not valid in expect.)
\r	Send or expect a carriage return.
\s	Represents a space character in the string. This may be used when it is not desirable to quote the strings which contains spaces. The sequence 'HI TIM' and HI\sTIM are the same.
\t	Send or expect a tab character.
\\	Send or expect a backslash character.
\ddd	Collapse the octal digits (ddd) into a single ASCII character and send that character. (some characters are not valid in expect.)
^C	Substitute the sequence with the control character represented by C. For example, the character DC1 (17) is shown as ^Q. (some characters are not valid in expect.)

Termination codes

The chat program will terminate with the following completion codes.

0	The normal termination of the program. This indicates that the script was executed without error to the normal conclusion.
1	One or more of the parameters are invalid or an expect string was too large for the internal buffers. This indicates that the program as not properly executed.
2	An error occurred during the execution of the program. This may be due to a read or write operation failing for some reason or chat receiving a signal such as SIGINT.
3	A timeout event occurred when there was an expect string without having a "-subsend" string. This may mean that you did not program the script correctly for the condition or that some unexpected event has occurred and the expected string could not be found.
4	The first string marked as an ABORT condition occurred.
5	The second string marked as an ABORT condition occurred.
6	The third string marked as an ABORT condition occurred.
7	The fourth string marked as an ABORT condition occurred.
...	The other termination codes are also strings marked as an ABORT condition.
	Using the termination code, it is possible to determine which event terminated the script. It is possible to decide if the string "BUSY" was received from the modem as opposed to "NO DIAL TONE". While the first event may be retried, the second will probably have little chance of succeeding during a retry.

Copyright

The chat program is in public domain. This is not the GNU public license.

4.6 *chmod* - Changes File Permissions

Synopsis

```
chmod <mode> <filename> ...
```

Description

Changes permission bits on the named files, in accordance with the specified mode. The mode may be specified as an octal absolute mode, or as a symbolic relative mode, as specified below.

Exactly how any requested set of permissions will be interpreted depends on the filesystem in question. For example, permissions may be ignored altogether, or there might be no distinction between read and execute permissions.

Symbolic modes

A symbolic mode is actually a list of instructions specifying how to modify the existing mode of the file. The grammar is character-based; no whitespace is permitted.

The mode consists of a comma-separated list of clauses. Each clause starts with a set specification, saying which set of permissions is to be affected. One or more of the following may be used:

- u: user permissions
- g: group permissions
- o: other permissions
- a: all of the above

If this specification is omitted, all permissions are affected, as if "a" had been specified.

After the set specification, there is a list of one or more actions, saying how to modify the specified permissions. Each action consists of an action character, followed by a bit specification. The action character must be one of:

- +: set the specified bits
- -: clear the specified bits
- =: set the specified bits and clear all others

The bit specification may be one of "u", "g" or "o", specifying whichever bits are set in the appropriate permission set. Otherwise it must be a list of zero or more of the following:

- r: permission to read
- w: permission to write
- x: permission to execute/search
- X: as "x" if any execute bits were set originally, or if the file is a directory; otherwise nothing
- s: set-user-ID (u set) or set-group-ID (g set)
- t: sticky bit (u set)

4.7 *cipher* - Encrypts Data

Synopsis

```
cipher <algorithm> <key>
```

Description

Reads for standard input, encrypting the data using the specified encryption algorithm and key, and sending the encrypted data to standard output.

The <*algorithm*> specified must be a class derived from *lib/cipher*. By default, "lib/cipher/" is prepended, so, for example, the DES algorithm, implemented in the class *lib/cipher/des*, can be specified as "des". If the

name specified starts with a "/", it is taken as an absolute class name. In any case, "/_new" will be appended to get the name of the allocator tool.

If the <*algorithm*> string contains "//", then the rest of the string is taken as an option string, and is passed to the selected cipher algorithm class. The interpretation varies from class to class; for example, to the null cipher class it specifies the block size. This argument defaults to an empty string, the interpretation of which is algorithm-dependent.

The <*key*> must consist of an even number of hexadecimal digits. This will be converted into a byte string, which will be passed to the cipher as a key value. Any length of key may be specified; the exact interpretation depends on the cipher algorithm.

The input will be padded with NUL bytes, if necessary, to make it an integral number of blocks in length.

Options

-d	Use the specified cipher algorithm in decryption mode, instead of encryption.

4.8 *ckmem* - Check intent™ Memory Structures

Synopsis

```
ckmem
```

Description

This function asks the intent memory allocator to check its internal data structures for validity. The system will be halted if it detects memory corruption.

UNIX equivalent	None applicable
DOS equivalent	`mem`

4.9 *cksum* - Perform Cyclic Redundancy Check

Synopsis

```
cksum [<filename> ...]
```

Description

This function calculates a 32-bit CRC (cyclic redundancy check) of the contents and length of each specified file (or standard input if no filenames have actually been specified). One line is output for each file processed, containing the CRC, the file's length in octets, and the filename.

This function is often used to make sure that any files that may have been transferred by unreliable means have not been corrupted in the process. It works by comparing the *cksum* output for the received files with the *cksum* output for the original files.

The CRC used is that specified by the Ethernet standard, and mandated for this utility by POSIX.2. The CRC is not cryptographically secure; it is feasible to deliberately generate a file with any desired CRC value. *md5sum* gives a higher degree of both error detection and cryptographic security.

UNIX equivalent	cksum
DOS equivalent	None applicable

4.10 *cls* - Clear Screen

Synopsis

```
cls
```

Description

Sends the clear screen sequence (ESC E) to stdout.

UNIX equivalent	clear
DOS equivalent	`cls`

4.11 *cmp* - Compare Files

Synopsis

```
cmp <file1> <file2>
```

Description

The two specified files are compared, byte by byte. The exit status is 0 if the files are identical, or 1 if they differ. If the files are identical, no output is generated. By default, if the files differ, the point where they differed will be displayed on standard output, and if one file is shorter than the other then this it will be indicated on standard error.

Example

```
cmp home/default/shell.rc home/users
```

UNIX equivalent	`cmp`
DOS equivalent	None Applicable

Options

-l	Output a list of all differing bytes.
-s	Suppress all output.

4.12 *comm* - Outputs File Lines In Three Columns.

Synopsis

```
comm <file1> <file2>
```

Description

The lines within each of the two specified files must be in sorted order (as if sorted by sort in default mode). *comm* outputs the lines of the two files in three columns: the first column contains the lines that appear only in the *<file1>*, the second column contains the lines that appear only in the *<file2>*, and the third column contains the lines that appear in both files.

Options

-1	Do not output the first column (lines appearing only in <file1>).
-2	Do not output the second column (lines appearing only in <file2>).
-3	Do not output the third column (lines appearing in both files).

4.13 Counttools

Determine the size of tools from an image map file.

Synopsis

```
counttools [<options>] <map file> <search pattern>
```

Description

This script analyses a map file produced when a system image is built to determine the size of a specified collection of tools.

This is a utility for development use to assist in determining what the code size is for different components of the system. The parameter <map file> gives the map file to use. If no extension is specified then '.map' will automatically be added.

The tools that are included are determined by <search pattern>. This is a pattern, which is matched to the tool filename (note that this is not necessarily the actual tool name).

Options

Only one option is currently defined:

-n	This option suppresses the trailing carriage return. This can be useful when it is called from another script.

Examples

The following command determines the size of the tools in the lib hierarchy in the map file *sys/platform/win32/pjavafull.map.*

```
counttools sys/platform/win32/pjavafull lib
```

4.14 *continue* - Jump to Top of Inner-Most Loop

Synopsis

```
continue
```

Description

This command simply causes the execution of the *body* statement of the inner-most loop to finish, returning a status of zero. When used in a while loop, the *while* condition will be re-evaluated before executing the body statement of the loop again.

4.15 *cp* - Copy Files

Synopsis

```
cp <source> ... <directory>
cp <source> <destination>
```

Description

This function copies the files as specified by the source arguments. If the last argument is a directory, all the source files are copied to names which are within the target directory and have the same basename as the source argument. Otherwise there must be exactly two arguments, and the first one is copied to the second.

Destination files that already exist will be overwritten. Destination files that do not already exist will be created with the same permission flags as the source file.

UNIX equivalent	cp
DOS equivalent	copy

Options

-f	Delete unwritable destination files. Without this option, unwritable destinations will not be overwritten.
-i	Query the user before overwriting any file.
-p	Attempt to duplicate each file's timestamps, ownership and permission bits.
-r	Copy directories recursively. Without this option, directories are not copied.
-R	A POSIX-mandated synonym for -r.
-u	Only copy files which do not exist in the destination or where the source file has been modified more recently than the destination file.
-v	Print the names of all files copied

4.16 *csum* - Calculates Checksums.

Synopsis

```
csum [<filename> ...]
```

Description

Calculates checksums of the specified files (standard input if no filenames are specified). One line is output for each file processed, containing the textual representation of the checksum, and the filename. The checksum algorithm to be used must be specified, using the -a option.

Options

-a <algorithm>	Use the specified checksum algorithm. (Actually, most of the algorithms in current use are not, strictly speaking, checksums. However, no other term has such wide currency.) The algorithm specified must be a class derived from *lib/cksum*. By default, "lib/cksum/" is prepended, so, for example, the MD5 message digest algorithm, implemented in the class *lib/cksum/md5*, can be specified by an option "-amd5". If the name specified starts with a "/", it is taken as an absolute class name. In any case, "/_new" will be appended to get the name of the allocator tool. If the <*algorithm*> string contains "//", then the rest of the string is taken as an option string, and is passed to the selected checksum algorithm class. The interpretation varies from class to class; for example, to the CRC class it specifies the generator polynomial to be used. This argument defaults to an empty string, the interpretation of which is algorithm-dependent.
-r	Output the raw binary form of each checksum, instead of a textual representation. Filenames and formatting are suppressed.
-s	Checksum the command line arguments, as strings, rather than treating them as filenames. At least one argument must be given.
-v	Before generating any checksums, list the names of the selected algorithms on standard output. When the checksums are being saved for long-term storage, this reduces the risk of forgetting which algorithm was used to generate a set of checksums.

4.17 *cut* – Concatenate Files and Send Part of Input Line to Stdout.

Synopsis

```
cut [<filename> ...]
```

Description

Concatenates all the named files, or reads standard input if none are specified. Part of each input line, selected in accordance with options specified, is copied to the standard output.

Options

-b <list>	Output the specified bytes of each input line. The list is a comma- or blank-separated list of byte ranges. Each byte range is a dash-separated pair of decimal numbers representing an inclusive range (the first byte on each line is number 1). Either endpoint of a range may be omitted to make the range open-ended; A single number may be used alone to represent a range of length one. Any range specified beyond the end of the line is ignored.
-c <list>	Output the specified characters of each input line. The list is specified as for -b.
-d <character>	Sets the field separator (delimiter) to the specified character.
-f <list>	Output the specified fields of each input line. The list is specified as for -b. Fields are considered to be separated by each instance of the delimiter character (by default the tab character). By default, lines containing no instances of the delimiter character are copied to the output in their entirety.
-n	Do not split in the middle of characters when using -c.
-s	When using -f, suppress lines containing no instances of the delimiter character, rather than

	copying them to the output unchanged.

5. Commands D

5.1 *date* - Display Date

Synopsis

```
date [+<format>] [-a] [-s<datestring>] [-u] [-R]
```

Description

This function displays the current date and time.

The time format is `[[[[CC]YY]MM]DD]HHMM[.SS]` (as per ISO 8601:1988) which describes Century, Year, Month, Day, Hour, Minutes, Seconds respectively. Each field is padded with leading '0' characters if required to bring the field width to the required 2 digits. The length of the string is used to determine the format. The hours and minutes fields are required in 24-hour clock format, with all other fields being optional. If a field is undefined then the current value is used for that field. eg, if the century is undefined, then the current century is retained. The exception is seconds - if these are not defined, then 00 is used.

Years from 0000 to 9999 are valid, however the host system may have a reduced range, causing any "set" to fail.

UNIX equivalent	`date`
DOS equivalent	`date/time`

Options

+<format>	A string following a '+' is treated as a *strftime* format string. If not defined, the format will default to "%a %b %e %H:%M:%S %Z %Y". If the leading '+' is missing, the string is treated as a time setting string. See "Time Format" in the "-s" section below.
-u	Set or display the time as UTC. This option can be used with the -a and -s options.
-a<[+/-]seconds>	Adjust the time by +/- seconds. The clock is read, adjusted by the seconds specified, and written back.
-R	Display the time in RFC format. eg "%a, %e %b %Y %H:%M:%S %z"
-s<new time>	Set the clock to the new time.

5.2 *dbgbp* - Cause Breakpoint

Synopsis

```
dbgbp
```

Description

This function causes a breakpoint by calling the tool *sys/cii/breakpt*. This is normally used to enter a debugging mode of some kind.

UNIX equivalent	None Applicable
DOS equivalent	None Applicable

5.3 *dd* - Converts a file whilst copying it.

Synopsis

```
dd [<arg> ...]
```

Description

By default, dd copies its standard input to its standard output. It does all I/O in blocks of a predetermined size, by default 512 bytes. When it has finished, a message is sent to standard error, indicating how many complete and partial blocks were input and output.

dd is designed to operate correctly and usefully on a wide range of devices, including disk and tape devices, as well as regular files. Some of the behaviour is defined with such usage in mind.

Options

The arguments act as options, controlling aspects of dd's behaviour:

if=<filename>	Use the specified file for input, instead of standard input.
of=<filename>	Use the specified file for output, instead of standard output.
ibs=<blksize>	Perform input in blocks of the specified size (default 512). The blocksize is specified as a number of bytes. It may be suffixed by "b" to multiply by 512, or "k" to multiply by 1024. Two or more numbers may be given, separated by "x", to multiply them.
obs=<blksize>	Perform output in blocks of the specified size (default 512). The blocksize may be specified as for ibs.
bs=<blksize>	Perform input in blocks of the specified size, but do not collect up output into complete blocks - instead, output each block processed immediately, even if it is a partial block. The blocksize may be specified as for ibs. This option overrides ibs and obs.
skip=<number>	Skip over the specified number of input blocks before performing normal processing. A seek will be performed if possible, but this option is most useful on non-seekable devices, where the specified number of blocks will be read and discarded.
seek=<number>	Skip over the specified number of output blocks before performing normal processing. A seek will be performed if possible, but this option is most useful on non-seekable devices, where the specified number of blocks will be read and discarded.
count=<number>	Stop after processing the specified number of input blocks.
conv=<options>	<options> is a comma-separated list of words, referring to boolean options: • lcase - Any byte in the input that is an uppercase character is converted to lowercase. • ucase - Any byte in the input that is a lowercase character is converted to uppercase. • swab - Swap each pair of adjacent bytes. • swas - Swap each pair of adjacent shorts (two-byte blocks). • swai - Swap each pair of adjacent words (four-byte blocks). • swal - Swap each pair of adjacent longwords (eight-byte blocks). • noerror - Continue processing after input errors. An attempt will be made to seek to the • next input block. • notrunc - Do not truncate the output file. By default, an attempt is made to truncate the • output file, after any seek option has been processed. • sync - If a partial input block is read, it is padded with NULs before being processed. When noerror is also used, this option will fill a completely unreadable block with NULs, whereas the block would otherwise be dropped.

5.4 Depend - Displays dependencies for tools.

Synopsis

```
depend [ <options> ] <toolname>
```

Description

This utility provides a list of dependencies on other tools for the tool specified. The dependencies are calculated recursively, up to the depth specified, and an indented list is displayed. Tools whose names are followed by * are duplicated in the list.

Options

-o	(ordered): display alphabetically sorted list (removes duplicates)
-n <depth>	only find first <depth> dependencies (default maximum depth)
-c	(count): display reference count (ordered list only)
-q	(quiet): do not display dependencies (normally used with -s)
-s	(summary): display summary
-t	(translate): use specified translator to find additional dependencies
-e	(extension): use specified tool extension list
-v	(virtual): disregard virtually referenced dependencies
-f	(fixup): disregard fixup, virtual+fixup and virtual dependencies
-h	(help): display help message

5.5 *defun* - Define Shell Function

Synopsis

```
defun <function> <body>
```

Description

This defines a shell function with the name and body that have been specified. This function will then behave like a normal command.

When it is executed, the body of the function is executed within the current shell. The name of the function is temporarily stored in the shell variable *shell.argv0*, and any arguments in *shell.argv*. The function exits with the status of the last command in its body, but execution can be terminated early by using the return command.

Example

```
defun x {echo hello}
```

This defines the function "x" with the above line, and enters "x" at the shell prompt, so that whatever command(s) are between the {} will the be executed. In this case, "hello" will be echoed.

UNIX equivalent	function
DOS equivalent	None Applicable

5.6 *devinfo* - Obtain Device Driver Information

Synopsis

```
devinfo [ <pathname> ... ]
```

Description

For an application to be able to use a Device Driver, it must first be mounted. All the procedures place a reference to the Device Driver in a look-up table called a Mount List and a *<name>* which is associated with this reference.

If no arguments are specified, the *devinfo* function will display this mount list on standard output. If arguments are also specified, information will be output concerning the device mounted at each pathname. Further information concerning Device Drivers can also be found in "*The Device Driver Design Guide*".

Example

```
devinfo dev/_loadisr
```

UNIX equivalent	`mount`
DOS equivalent	None applicable

5.7 *devstart* - Load Device Driver

Synopsis

```
devstart <mount-point> <class-name> [ <args> ... ]
```

Description

There are two distinct means by which a device driver may be loaded into memory. It is possible to use either the sysgen function *.obj*, or the shell function *devstart*.

- *devstart* is used to load device drivers in a running system. (To load device drivers from the *sysgen* steering file, use the *.obj* directive.)
- <*mount-point*> is the pathname to be given to this object instance. This pathname will be added to the mount table and the object instance pointer will be associated with it. The pathname conventionally starts with "/device/", but this is not required. When the object is mounted, it will be accessible as a part of the filesystem, starting at this point.
- <*class-name*> is the name of the class of the object to be mounted. "/_new" will be appended to generate the name of the allocation tool.
- <*args*> are additional arguments which are passed to the device initialisation method. Their meaning is dependent on the device type, but will usually take the form of a command line, with options. Note that if there are any options to the device, or any other type of argument starting with "-", it will be necessary to use a "--" argument to terminate *devstart*'s options.

UNIX equivalent	`mount`
DOS equivalent	No exact equivalent - function performed by `autoexec.bat` and `config.sys`

Options

-l	Mount the device locally only; i.e., do not make it visible to other networked processors.
-d	Mark the mount point as being associated with the specified device, but do not initialise the device until it is referenced.

5.8 *devstop* - Remove Device Driver

Synopsis

```
devstop <pathname> ...
```

Description

This function is used to remove any specified device driver from the system, by unmounting the device at each pathname. The Mount List is searched for the specified <*name*>. If it is found the object instance is deleted, and the <*name*> is removed from the Mount List. More detailed information concerning this function can be found in the "*Elate Device Driver Design Guide*".

UNIX equivalent	`unmount`
DOS equivalent	No exact equivalent - function performed by `autoexec.bat` and `config.sys`

5.9 *df* - Summarise File System Information.

Synopsis

```
df [<device_name>]
```

Description

Outputs information on the file system requested. If no <device_name> is specified then all mounted file system devices are displayed.

Devices flagged for delayed mounting are not displayed.

The information displayed under "Filesystem" is simply a truncated form of the device information string.

5.10 *diff* - Produce List of Differences Between Compared Files

Synopsis

```
diff <file0> <file1>
```

Description

Compares the named files, sending a list of differences to standard output. The exit status is zero if no differences are found, 1 if differences are found, or 2 on error.

If both named files are directories, a directory comparison is performed as described below. If exactly one of the named files is a directory, then the contents of the non-directory is compared with the contents of the file in the directory with the same basename as the non-directory. Otherwise the contents of the named files are compared directly.

Example

```
diff demo/ave/blend.asm demo/ave/blendimage.asm
```

Directory comparison

When two directories are being compared, the files in them are processed in alphabetical order. For each file appearing in exactly one of the directories, an informative message is output. Regular files appearing in both directories have their contents compared. Non-regular files are not compared, and an informative message is output.

Ignorable differences

Some options cause certain types of differences between files to be `ignored'. What this means is that the ignored type of change, alone, will not cause the files to be considered different (for the purposes of the exit status), and the change will not be output if it can be avoided.

When a change is excised from the output *diff*, the *diff* will be modified to remain self-consistent. Line numbers specified for the second file will not necessarily refer to the actual second file, but will refer to a version of the second file which is identical to the first file in the area of the ignored changes.

If an ignorable difference is sufficiently close to a non-ignorable difference that they would appear in the same hunk of output, that hunk will be output in its entirety, including the ignorable change. When this occurs, that change is regarded as being part of the second file for the purposes of the rest of the *diff*.

Options

-a	Ignored. (In other versions of *diff*, this forces comparison of all files, whereas non-text files would not normally be compared. This version compares all files anyway.)
-b	Ignore changes of the amount or type of whitespace at any point on any line. The presence or absence of whitespace is still considered significant, except at the end of a line.
-B	Ignore changes that merely insert or delete blank lines.
-c	Use context *diff* output format, with 3 lines of context.
-C <lines>	Use context *diff* output format, with the specified number of lines of context.
-d	Ignored. (In other versions of *diff*, this requests that diff make extra effort to produce a minimal change set.)
-D <symbol>	Use *#ifdef* output format. All lines of both files being compared are output, with

	interspersed C preprocessor directives to select which file the output compiles as. This will not handle changes to preprocessor directives cleverly.
-e	Use the ed script output format.
-E	Use the forward ed script output format. Equivalent to "-lsilent -ofwded -Ochanges". The output is an ed script, that can be used as input to ed to change the first file into the second. The changes appear in the script in forward order, which means that the line numbers giving locations correspond to the second file, but the endpoints of ranges are not independently meaningful.
-f	Use the historical `forward ed script' output format. The output is ambiguous, and cannot be processed by ed or patch. It is only provided for historical compatibility.
-h	Ignored, for historical compatibility.
-H	Ignored. (In other versions of *diff*, this requests that *diff* use a modified algorithm that is particularly efficient for comparing large files with many small, widely separated changes.)
-i	Ignore changes in the case of any character.
-I <regexp>	Ignore changes that merely insert or delete lines matching the specified POSIX Basic Regular Expression.
-l <class>	Specifies the format to use for label output. The <*class*> argument is the name of an int**e**nt class, derived from *app/diff/ohdr*. By default, "app/diff/ohdr/" is prepended, so, for example, the unified *diff* header format, implemented in the class *app/diff/ohdr/unified*, can be specified by an option "-lunified". If the name specified starts with a "/", it is taken as an absolute class name. In any case, "/_new" will be appended to get the name of the allocator tool. If the <*class*> string contains "//", then the rest of the string is taken as an option string, and is passed to the selected class. The interpretation varies from class to class. This argument defaults to an empty string, the interpretation of which is class-dependent. The -l, -o and -O options, together, entirely determine the manner in which file differences are displayed. They can be used together to invent a new output format, or they can be used after one of the standard format options to modify that format. The default output format is equivalent to "-lsilent -onormal -Ochanges".
-L <label>	In output formats that output labels for the files being compared, this option can be used to override the default labels (which consist of filenames and timestamps). Up to two of these options may be specified, to override the first or both labels.
-n	Use the RCS *diff* output format.
-N	In a directory comparison, if a regular file exists in one directory and not in the other, perform a comparison as if there were a zero-length file in place of the non-existent file.
-o <class>	Specifies the format in which to output each hunk of differences. This is specified as an int**e**nt class derived from *app/diff/ohunk*, in the manner of the -l option, except that the automatic prefix is "app/diff/ohunk/".
-O <class>	Specifies the algorithm to use to gather nearby changes together into hunks. This is specified as an int**e**nt class derived from *app/diff/odiff*, in the manner of the -l option, except that the automatic prefix is "app/diff/odiff/".
-P	In a directory comparison, if a regular file exists in the second directory and not in the first, perform a comparison as if there were a zero-length file in place of the non-existent file. This is like -N, except that it only applies in one direction.
-q	Do not output differences; this can be used if only the exit status is required. Usually, however, it is preferable to use the *cmp* utility.
-r	Recursively examine directory trees. Specifically, in a directory comparison, if both directories have subdirectories of the same name, a directory comparison is performed on the subdirectories. If a subdirectory exists in only one of the directories, it is not compared against anything.
-s	Report, on standard output, when files whose contents are compared turn out to be identical.
-u	Use unified *diff* output format, with 3 lines of context.
-U <lines>	Use unified *diff* output format, with the specified number of lines of context.
-w	Ignore changes that add, remove or change whitespace characters other than newline. This takes precedence over -b.
-X <filename>	In a directory comparison, ignore any files whose basenames match any of the whitespace-separated glob patterns in the specified file.
-x <pattern>	In a directory comparison, ignore any files whose basenames match the specified glob

	pattern.

5.11 *dir* - List Named Files

Synopsis

```
dir [<filename> ...]
```

Description

Lists the named files. If a directory is named, its contents are listed. If no filenames are given, the contents of the current directory are listed. *dir* is actually *ls* in disguise - its options and defaults approximate the MS-DOS *dir*, but the output is always actually produced by *ls*. Please note that (as is standard for int**e**nt) options are introduced by "-", rather than the DOS "/".

Example

```
dir -s dev
```

UNIX equivalent	`ls`
DOS equivalent	`dir`

Options

-a <anything>	Show all files in a directory (ls -a). By default files whose names begin with "." are hidden.
-b	Output filenames one per line (ls -1). This takes precedence over -w.
-l	Ignored, for DOS compatibility. (In DOS it forces the monocase filenames to be displayed in lowercase.)
-o <letters>	Controls the sorting of files. A ":" as the first letter after the option is ignored. The sorting is determined by the first specified letter: • a By last access time, least recent first (ls -rut). • d By last modification time, least recent first (ls -rt). • e Alphabetically by extension (ls -X). • g Ignored for DOS compatibility. • n Alphabetically by filename. (Default behaviour in both ls and dir.) • s By size, smallest first. A "-" before the letter reverses the sort order. (Equivalent to toggling the use of ls's -r option.)
-p	Ignored, for DOS compatibility. (In DOS it causes the output to be paged. To get the equivalent effect, filter the output through more.)
-s	Recursively list directories encountered (ls -R).
-v	Produce a more verbose listing (ls -s).
-w	Produce a multi-column "wide" listing, showing only filenames (ls -C). By default files are listed one per line, with more information (ls -l).

5.12 *dirname* - Strip Non-Directory Suffix From Filename

Synopsis

```
dirname <pathname>
```

Description

This function outputs, terminated by a newline, all but the last component of the specified pathname.

Example

```
dirname x/y/z.c
```

This outputs to stdout "x/y", thereby only printing out the directory path.

UNIX equivalent	`dirname`
DOS equivalent	None Applicable

5.13 *display* - Format Text Displays

The *display* command is used to supplement the functionality provided by the Shell Line Editor. It is primarily used by the editor, to display the line being edited.

Synopsis

```
display f <var> [<command> ...]
```

```
display <old-state> <new-state>
```

Description

This command is used to maintain text displays.

The first synopsis, with the argument "f", is used to format specified text for display on a terminal. The command arguments specify how to build up the display, in the manner described here.

Specification Language

In the first synopsis form, a sequence of commands is used to specify an appearance of a display. Multiple commands may be specified in each command line argument, without separator, except where a command takes an argument that stretches to the end of the command line argument.

Commands are:

ab<c>	Set the background colour to that specified by the hexadecimal digit <*c*>. Each bit of the value of the hexadecimal digit has a distinct meaning: bit 0 is red, bit 1 green, bit 2 blue, and bit 3 bright. The initial value is 0 (black).
af<c>	Set the foreground colour to that specified by the hexadecimal digit <*c*>. This is interpreted in the same way as for the *ab* command. The initial value is 7 (white).
c	The cursor goes here. If there is no such command in the array, the cursor will be hidden if possible. If there is more than one cursor command in the array, the result is unspecified.
n	Newline: move to the beginning of the next line down. (A newline in a text sequence displays as "^J".)
t<text>	Output the specified text (which continues to the end of the argument). Unprintable characters are converted to a printable representation. If the end of the physical line is reached, the text is wrapped.
Cs<options>	Start a section of columnar display, the top line of which appears on the current line. The options continue to the end of the argument, and currently must be empty. Between this and the corresponding *Ce* command must be a sequence of zero or more *Ct* sections.
Ce	End a columnar display started with *Cs*, and move to the beginning of the last line of the columnar display.
Ct	Start an entry in a columnar display. Typically this will be followed by a *t* command giving text to display, but other commands may be used too. Each such entry is displayed in the same sized area, big enough for any entry. This command must be followed by the normal commands defining the contents of the column entry. Any attribute changes are local to the entry.

The resulting formatted display is stored in the specified shell variable. The form of this formatted display is unspecified, except that it is at least two characters long. (in particular it should be noted that it may include NULs and other unprintable characters.) This value can only be meaningfully used as an argument to the second synopsis of this command.

Example

This example shows how the first synopsis of the display command is used by the Shell Line Editor.

```
display -- f sle.ndisp $sle.prompt t$sle.lline {ct^}
```

The second synopsis outputs text and control codes to change a physical display from that specified by <*old-state*> to that specified by <*new-state*>. The arguments may take the following special values:

n	Terminal in uninitialised state, screen contents garbage, cursor on top line of display area.
d	Terminal uninitialised, cursor at start of the line below the display.

See also - *parse*.

6. Commands E

6.1 *echo* - Copy Arguments To Standard Output

Synopsis

```
echo [<arg> ...]
```

Description

This function copies its arguments in turn to standard output, separated by spaces and followed by a newline.

UNIX equivalent	echo
DOS equivalent	echo

Options

-n	Suppress the final newline.

6.2 *egrep* - Print Lines Matching Pattern

Synopsis

```
egrep <expression> [<filename> ...]
```

Description

The function *grep* (see below) is used to display all of the lines in any named files that match a given pattern. *egrep* acts exactly like *grep*, except that it defaults to interpreting the expressions as Extended Regular Expressions, as if the -E option were specified.

UNIX equivalent	egrep
DOS equivalent	None Applicable

6.3 Ejcode - Runs a Java™ application.

Synopsis

ejcode *<classname> [<argument> ...]*

Description

Ejcode starts a Java™ virtual machine (JVM) and attempts to run a Java application in it. Execution starts in the main method of <classname>. The <argument>s are passed to the main method in an array of java.lang.String objects.

If no arguments are supplied, a NULL reference is passed to the main method. The exit status of this command is not defined.

This command differs from the standard jcode utility in that it is intended for use in embedded targets. Consequently code size is minimised by omitting any diagnostic strings or other error messages. It is expected that the user of this utility has configured the Java™ virtual machine (JVM) by other means to set up heap size, stack size, verification mode, etc.

Options

-e	Cause Elate to exit (by executing sys/pii/shutdown) once the Java™ virtual machine (JVM) has exited.

6.4 Entropyctl - Controls the kernel entropy collector.

Synopsis

```
entropyctl <command> ...
```

Description

Entropyctl provides a shell interface to control operations on the kernel entropy collector (sys/kn/entropy). Each command line argument is taken as a command from the list below.

Commands

status	List the same information as vstatus, but in a machine-readable form.
vstatus	List the current status of the entropy collector to standard output. The information listed is the entropy pool size, current entropy count, and the total minimum and maximum sizes of all pending read operations. All these values are in bits.

6.5 *env* - Writes Environment to *stdout*

Synopsis

```
env
```

Description

Writes its environment to standard output, one entry per line. However, no quoting is done, so the output is ambiguous; it is usually preferable to use the shell's own mechanisms to display parts of the environment. Typing "env" will give all of the environment variables, including the shell scripts. Variables can be set using the 'set' command, e.g. *set var*. That variable may then be displayed via the *echo* command, e.g. *echo $ var.*

UNIX® equivalent	*env*
DOS® equivalent	None Applicable

6.6 *eval* - Execute Command and Return Exit Status

Synopsis

```
eval <command>
```

Description

This executes the specified command in the current shell environment. *eval* then returns the exit status of the last command executed, or zero if there were no non-null commands.

This is what the shell does if invoked using the -c option.

Example

```
eval ps
```

UNIX equivalent	`eval`
DOS equivalent	None Applicable

Options

-S <status>	Return the specified status, instead of zero, if there are no non-null commands. This option has only specialised use, in scripts trying to simulate the execution of a normal command sequence.
-c <variable>	If the command exits via an exception, catch the exception, and save it in the

	specified variable. Otherwise set the variable to an empty array value. If this option is not used, any exception will propagate out past the *eval*.
-s <variable>	Save the command's exit status in the specified variable.

6.7 *exit* - Leave int**e**nt Shell

Synopsis

```
exit [<status>]
```

Description

This causes the current shell to terminate with the specified status (default zero). it is strongly recommended that either this function or shutdown should be used to leave int**e**nt, rather than by simply deactivating the int**e**nt session.

UNIX equivalent	exit
DOS equivalent	exit

6.8 *expand* - Expands TAB Characters.

Synopsis

```
expand
```

Description

Copies standard input to standard output, expanding each TAB character to the appropriate number of spaces to have the same visual effect.

Options

-t <tablist>	Specifies where tabstops are located. <tablist> is a whitespace or comma-separated list of decimal column numbers. If only one number is given, tabstops are separated by that many columns (default 8). If more than one number is given, tabstops are at the named columns, and every column after the last specified. Tabstops must be listed in increasing order, and column zero is not valid.

7. Commands F

7.1 *false* - Does Nothing

Synopsis

```
false [<arg> ...]
```

Description

This function does nothing, ignores any arguments, and exits with a status of 1 (which conventionally indicates failure). *false* is commonly used used as a place holder in shell scripts where an unsuccessful command may be required.

UNIX equivalent	`false`
DOS equivalent	None Applicable

7.2 *fgrep* - Display All of the Lines in any Named File Matching a Given Pattern

Synopsis

```
fgrep <expression> [<filename> ...]
```

Description

The function *grep* is used to display all of the lines in any named files that match a given pattern. *fgrep* acts exactly like *grep*, except that it defaults to interpreting the expressions as fixed strings, as if the -F option had been specified.

UNIX equivalent	`fgrep`
DOS equivalent	None Applicable

7.3 *file* - Determines the Type of a File.

Synopsis

```
file <pathname> ...
```

Description

file determines what type of file each specified pathname refers to, and writes this information on standard output. For regular files, the contents is examined, and *file* makes a guess at the type of data contained in the file. All reasonable guesses are output.

If a file cannot be examined, it is not treated as an error. An error message appears on standard output, as part of the normal output stream. The exit status is unaffected.

Options

-m <magic-file>	Specifies a non-default file type data file. See the *lib/filetype* documentation for details of the interpretation of the filename.

7.4 *find* - Find Files

Synopsis

```
find [<pathname> ...] <expression>
```

Description

This function searches for files matching the specified expression. The search is recursive, either starting at the specified pathnames, or from the current directory by default.

UNIX equivalent	`find`
DOS equivalent	None Applicable

Expressions

Each expression is evaluated for each file as it is processed. The expression evaluates to true or false; however, the final value of the expression is ignored. Sub-expressions are evaluated left-to-right, and their results are used to short-circuit evaluation, making the conditional execution of expressions with side-effects possible.

(<expression>)

True if the sub-expression is true. (This is used for grouping.) Note that parentheses will need to be quoted in the shell.

! <expression>

True if the sub-expression is false. Note that the exclamation mark will need to be quoted in the shell.

<expression-1> [-a] <expression-2>

True if both sub-expressions return true.

<expression-1> -o <expression-2>

True if either sub-expression returns true.

Predicates

-amin <n>	True if the file access time of the current file subtracted from the initialisation time is *n-1* to *n* minutes. The initialisation time is the start time of the *find* utility.
-anewer <file>	True if the access time of the current file is more recent than the modification time of the file named by the pathname <*file*>.
-atime <n>	True if the file access time of the current file subtracted from the initialisation time is *n-1* to *n* multiples of 24 hours. The initialisation time is the start time of the *find* utility.
-cmin <n>	True if the file creation time of the current file subtracted from the initialisation time is *n-1* to *n* minutes. The initialisation time is the start time of the *find* utility.
-cnewer <file>	True if the creation time of the current file is more recent than the modification time of the file named by the pathname <*file*>.
-ctime <n>	True if the file creation time of the current file subtracted from the initialisation time is *n-1* to *n* multiples of 24 hours. The initialisation time is the start time of the find utility.
-empty	File is empty and is either a regular file or a directory.
-false	Always evaluates to false.
-iname <pattern>	Like -*name*,(see below) but the match is case insensitive.
-ipath <pattern>	Like -path,(see below) but the match is case insensitive.
-iregex <pattern>	Like -regex, but the match is case insensitive.
-mmin <n>	True if the file modification time of the current file subtracted from the initialisation time is *n-1* to *n* minutes. The initialisation time is the start time of the *find* utility.
-mnewer <file>	True if the modification time of the current file is more recent than the modification time of the file named by the pathname <*file*>.
-mtime <n>	True if the file modification time of the current file subtracted from the initialisation time is *n-1* to *n* multiples of 24 hours. The initialisation time is the start time of the *find* utility.
-name <pattern>	True if the base of the filename matches the specified *glob* pattern. A leading "." can only be matched explicitly.
-newer <file>	True if the base of the filename matches the specified *glob* pattern. A leading "." can only be matched explicitly.
-path <pattern>	True if the entire pathname matches the specified *glob* pattern. "." and "/" are treated as

	ordinary characters.
-prune	If *-depth* is not in effect, evaluate to true, and do not process files within this directory. (No effect if the current file is not a directory.) If *-depth* is in effect, evaluate to false.
-regex <pattern>	True if the entire pathname matches the specified regular expression. (Note that this is not a substring match, as is common with regular expressions.)
-size <n[c]>	True if the file size in bytes, divided by 512 and rounded up to the next integer, is <*n*>. If n is followed by the character *c*, the size is specified in bytes, rather than in multiples of 512 bytes.
-true	Always evaluates to true.
-type <c>	True if the current file is of the type indicated by <*c*>: • b block special device c character special device • d directory • f regular file • l symbolic link (if supported) • p FIFO If the current file is a symbolic link, *-type* will by default apply to the link itself, but will look at the file it points to if *-follow* is used.
-xtype <c>	Like *-type*, but does the opposite type of stat (looks at the link if *-follow* is used, or the file pointed to otherwise).

Actions

Actions are really predicates with side-effects, and are treated internally as predicates. They return true except as specifically noted below.

-fprint <filename>	Print the pathname to the named file, followed by a newline.
-fprint0 <filename>	Print the pathname to the named file, followed by a NUL.
-print	Print the pathname to standard output, followed by a newline.
-print0	Print the pathname to standard output, followed by a NUL.

Options

it should be noted that *find* does not use the normal option convention. The options described below can be intermingled with parts of the expression, and must appear as separate arguments. If a pathname begins with `-', the only way to prevent it being treated as an option or part of the expression, is to prefix it with `./'.

-depth	Process each directory's contents before the directory itself. By default the directory is processed first.
-follow	Process the file a symbolic link points to, rather than the link itself.
-maxdepth<depth>	Do not process any files below the specified depth. Command line arguments are at depth 0.
-mindepth<depth>	Do not process any files above the specified depth. Command line arguments are at depth 0.
-noleaf	Ignored, for compatibility with Unix versions of *find*.

7.5 *fold* - Folds Long Lines.

Synopsis

```
fold [<file> ...]
```

Description

Copies the contents of the named files, or standard input if no files are specified, to standard output, folding long lines. Lines are folded, by inserting newline characters, such that they do not exceed a maximum number of columns, by default 80. Most characters count as taking up one column; backspace moves back one column, TAB moves forward to a multiple of 8 columns, and carriage return and newline return to column zero.

Options

-b	Treat the maximum width as a number of bytes, rather than columns, This has the effect of

	disabling the special handling of backspace, TAB and CR.
-w <width>	Specifies the maximum width, which must be a positive decimal number.

7.6 *for* - Repeatedly Execute Specified Command

Synopsis

```
for <var> [ <value> ... ] <command>
```

Description

This function repeatedly executes the specified command, with the specified variable set in turn to each of the specified values. The variable remains in all respects normal; it may be unset or changed in the loop, and will simply be set to the next value at the beginning of the next iteration.

The exit status is the exit status of the last iteration of the loop body. The variable is left with the value it had at the end of the last iteration of the loop body.

If no values are specified, the loop body is never executed, the exit status is zero, and the variable is unaffected. It is possible to use the *break* and *continue* commands to modify the flow of control through a *for* loop.

UNIX equivalent	`for`
DOS equivalent	`for`

intent Code	`for a 1 2 3 { ...command... }`
UNIX code	`for a in 1 2 3; do ...command...; done`

7.7 *ft* - Flush Specified Tools

Synopsis

```
ft [<toolname> ...]
```

Description

Flushes the named tools, or all unreferenced tools if none are specified.

UNIX equivalent	None Applicable
DOS equivalent	None Applicable

Options

-v	Report the status of each attempted specific tool flush.

7.8 *ftrace* - Change Destination of *tracef* Output

Synopsis

```
ftrace [<filename>]
```

Description

Changes the destination of *tracef* output to the specified file, defaulting to */device/display*. Otherwise traces are sent to whatever file or the device has been specified. Alternatively, *ftrace* takes one of the following options.

UNIX equivalent	None Applicable
DOS equivalent	None Applicable

Options

-d <mount-point>	Affect the specified device (default */device/trace*).

-k	Specifies traces will be converted to *ktraces*. Output destination depends on the configuration of *ktraces* on the platform. A filename must not be specified.

8. Commands G and H

8.1 *gc* - Garbage Collect Shell Data

Synopsis

```
gc [<threshold>]
```

Description

This function garbage collects the shell's internal data structures. This has no semantic effects, and would eventually be performed automatically by the shell in any case. Its sole use is to momentarily reduce memory footprint, which may occasionally be useful at certain points in a script.

If an argument is given, it is used to set the threshold for automatic garbage collection. The interpretation of this argument is implementation-dependent and may change without notice; however, an invalid argument will be ignored without complaint.

UNIX equivalent	None Applicable
DOS equivalent	None Applicable

Options

-n	Do not perform garbage collection. (However, setting the threshold may cause automatic garbage collection to be triggered in any case.)
-v	Emit to standard error a brief report of the results of the garbage collection, and the current criteria for automatic garbage collection.

8.2 *grep* - Display All of the Lines in Any Named Files Matching a Given Pattern.

Synopsis

```
grep [<expression>] [<filename> ...]
```

Description

The function *grep* is used to display all of the lines in any named files (or standard input if none are specified) that match a given pattern. By default the pattern is specified as a Basic Regular Expression (a pattern that describes a set of strings), as defined by ISO 9945.2 clause 2.8.3. Optionally, Extended Regular Expressions (clause 2.8.4) can be used instead.

The expression argument is used if and only if no -e or -f options were used. The expression specified is taken as a list of newline-separated regular expressions. The named files (or standard input if no files are specified) are searched for lines matching any of the regular expressions. By default, matching lines are written to standard output. Exit status is 0 if at least one matching line was found, 1 if no matching lines were found, or 2 if an error occurred.

UNIX equivalent	grep
DOS equivalent	None Applicable

Options

-B	Interpret the expressions as POSIX Basic Regular Expressions. This is the default for *grep*.
-b	Prefix each line displayed with its byte offset. The first line in each file is at offset 0.
-c	Instead of normal output, display a count of matching lines in each file.
-C [<num>][,[<num>]]	Set the number of lines of context to output when displaying a matching line. Either number may be omitted, in which case it defaults to zero. If only one

	number is given (with no comma) it is used for both leading and trailing context. Othewise, the first number is used for leading context, and the second for trailing context. By default, no context is output (-C0). When context lines are output, they are separated from any filename or other prefix by "-" instead of ":". If contexts overlap, they will be merged, with no line being output more than once.
-E	Interpret the expressions as POSIX Extended Regular Expressions. This is the default for *egrep*.
-e <expression>	Instead of taking the expression as an argument, use the expression specified here. The expression is taken as a newline-separated list of regular expressions. Multiple -e arguments can be used.
-F	Interpret the expressions as fixed strings. A line matches if it contains the string exactly, in consecutive bytes. This is the default for *fgrep*.
-f <filename>	Instead of taking the expression as an argument, read the expression from the specified file. The file contents must be a list of regular expressions, one per line. Multiple -f arguments can be used.
-h	Do not prefix output lines with filenames. By default, each output line is prefixed by the file it is taken from, if at least two files were specified on the command line.
-i	Perform all pattern matching case insensitively.
-L	Instead of normal output, list the names of all files containing no matching lines.
-l	Instead of normal output, list the names of all files containing at least one matching line.
-n	Prefix each line displayed with its line number. Lines are numbered consecutively, the first line in each file being numbered 1.
-q	Produce no output; return an exit status only.
-s	Suppress error messages for non-existent or unreadable files.
-v	Search for lines not matching any of the specified expressions, instead of normal matching lines.
-w	Only match lines where the substring matching the expression consists of a complete word or words.
-x	Only match lines that match the expression in their entirety. (By default any substring match is accepted.)

See Also

lib/rx

8.3 *head* - Output The First Part of Files

Synopsis

```
head [<filename> ...]
```

Description

This function displays the first part of each specified file. If no files are specified, standard input is used instead. The "first part" of a file is by default the first ten lines. If more than one filename is specified, the name of each file is displayed in a banner preceding the extract of the file.

UNIX equivalent	head
DOS equivalent	None Applicable

Options

<sign><number>	Equivalent to *-n<sign><number>*. This must be the first option on the command line.
-c [<sign>]<number>[<multiplier>]	Equivalent to -n, but the units are bytes rather than lines. The number may be suffixed by a multiplier character: "b" for 512-byte blocks, "k" for kilobytes, or "m" for megabytes.
-n [<sign>]<number>	If the sign is not present or is "-", display the specified number of lines from each file. If the sign is "+", display lines up to and including the specified line, counting lines in reverse. (*head -n+1* will therefore display the entire file.)

-q	Do not display filename banners.
-v	Display filename banners, even if only one file is being processed.

8.4 *help* - Find and Display Help Information

Synopsis

```
help [ <text> ]
```

Description

Searches for the specified text in the index file created by *helpidx* (in the file */etc/docn.idx*). If a match is found, the corresponding information in the documentation file is displayed.

UNIX® equivalent	`man`
DOS® equivalent	`help`

8.5 *helpidx* - Create Documentation Index

Synopsis

```
helpidx [<name> ...]
```

Description

Creates a list of files as follows, and creates a documentation index from them:

1. If no files are specified, the current directory and all subdirectories are searched for files whose names end in ".html". All of these files are added to the list.
2. If a specified name is a file, the file is added to the list
3. If a specified name is a directory, it and all of its subdirectories are searched for files whose names end in ".html". All of these files are added to the list.

Each file is scanned, and all name anchors are printed to stdout, along with the filename and the document title. Information about the progress of the operation is printed to stderr. The suggested usage is as follows:

```
helpidx > docn.idx
```

This creates an index file *docn.idx* in the current directory.

8.6 *html* - Scan HTML File

Synopsis

```
html <name> [<anchor>] [<options>]
```

Description

Scans the specified html file, and optionally display the data.

Options

-c	Abort if illegal html syntax is found
-n	Print all name anchors in the file
-t	Print document title

If neither of the -n or -t options are specified, the program will go into a user-interaction mode, where the user can scroll through the file.

9. Commands I and J

9.1 *if* - Execute If Command

Synopsis

```
if <if-cmd> [ <then-cmd> [ {
 <else-if-cmd> <then-cmd> } ... ]
 [<else-cmd>] ]
```

Description

This function executes the <*if-cmd*>. If that returns a zero (true) status, the following <*then-cmd*> (if any) will be executed. Otherwise, the <*else-if-cmd*>s (if any) are executed in sequence, until one returns a zero (true) status, whereupon its <*then-cmd*> will be executed. If none of the <*else-if-cmd*>s returns true, the <*else-cmd*> (if any) is executed. The status is the exit status of the last command that has been executed.

UNIX® equivalent	`if`
DOS® equivalent	`if`

9.2 img2cpm - Convert an image file to .cpm format.

Synopsis

```
img2cpm [-t <type>] [<filename> ...]
```

Description

This utility converts a graphic image file to .cpm format. The names of the files to be converted should be given on the command line. If no <filename> arguments are specified, img2cpm will read a list of file names to convert from its standard input; these should be separated by newlines.

The format of the input file is identified based purely on the filename and filename extension. The set of file formats supported is documented in the intent media programming guide. The output file will have the same name as the input file, except that the existing extension will be replaced by a .cpm extension.

Options

-h | -help | -?	Print usage information.
-t <*type*>	Specify the pixel map type (default is 32)

9.3 *interact* - Read Shell Commands Interactively

Synopsis

```
interact
```

Description

This function commands the current shell environment to interactively read and execute commands. Commands are read from *interact*'s standard input (file descriptor 0), and prompts are displayed on standard error (file descriptor 2). If a command read causes any errors such as a parse error, expansion error or redirection error, only that command will be aborted - another prompt will be issued and the next command will be read and executed as usual. *interact* returns the exit status of the last command executed, or zero if there were no non-null commands.

This is what the shell does if invoked with no explicitly specified source of commands.

UNIX equivalent	None Applicable
DOS equivalent	None Applicable

9.4 *java* - Runs a Java™ Application.

Synopsis

```
java [options] ... <classname> [ <argument> ... ]
```

Description

java is a wrapper script for the *jcode* command which performs pre-processing on the command line.

At the moment only the classpath option, if specified, is affected. If any *.zip* or *.jar* files are in the classpath then these will be mounted using the *dev/fs/zip* device drive and the classpath modified. When the *jcode* command terminates these devices will be stopped.

The exit status of this command is the exit status of the Java program (as given as a parameter to *java.lang.System::exit()* and 0 otherwise.

9.5 Javac - Compiles a Java™ source file.

Synopsis

```
javac <file>
```

Description

javac runs the java compiler. int**e**nt JTE does not provide a java compiler. If you want to use the Java compiler supplied with JDK 1.1 you should follow the instructions below to extract the compiler classes onto int**e**nt.

Options

See the documentation for the javac compiler supplied with the JDK.

Enabling javac on intent JTE.

JDK 1.1 includes a java compiler written in the Java language. This will run on int**e**nt JTE like any other conformant Java application. JDK 1.1 can be downloaded from JDK 1.1. It includes a a zip file, named classes.zip, containing all the java classes supplied with the JDK. Copy this file to the root of the int**e**nt files system and run the following sequence of shell commands at the int**e**nt shell.

- unzip -d /app/java/javac /classes.zip {sun/io/*}
- unzip -d /app/java/javac /classes.zip {sun/tools/asm/*}
- unzip -d /app/java/javac /classes.zip {sun/tools/java/*}
- unzip -d /app/java/javac /classes.zip {sun/tools/javac/*}
- unzip -d /app/java/javac /classes.zip {sun/tools/tree/*}

9.6 *jcode* - Runs a Java™ Application.

Synopsis

```
jcode <classname> [ <argument> ... ]
```

Description

jcode starts a Java™ virtual machine (JVM) and attempts to run a Java application in it. Execution starts in the main method of <*classname*>. The <*argument*>s are passed to the main method in an array of *java.lang.String* objects.

The exit status of this command is the exit status of the Java program (as given as a parameter to *java.lang.System::exit()* and 0 otherwise.

Options

Note that the options for this command differ from other intent shell commands - they have long names, and they must appear before <*classname*>. Anything after <*classname*> is taken to be an <*argument*>, even if it starts with a hyphen character.

-verify	Verify all classes when read in. For this to work, the system must be configured with a jcode translator which is capable of verification.
-verifyremote	Verify only classes which are loaded by a custom class loader. For this to work, the system must be configured with a jcode translator which is capable of verification.
-noverify	Do not verify any class. This is the default.

For more information upon the options for this particular command, please see *app/stdio/jcode.html.*

9.7 *jobs* - Display Current Shell Tasks

Synopsis

```
jobs [<id> ...]
```

Description

This function displays the status of shell 'jobs'. The shell maintains a list of jobs that are executed asynchronously. Every job is composed of zero or more `subjobs', each of which corresponds to a single process.

Each job has a unique ID number, which is a positive integer; job IDs are reused. Similarly, each subjob within a job has an ID number. A subjob can be referred to uniquely by using a combination of its job ID and subjob ID. Where a job ID is specified to a command, it is specified in the form *j*<*id*>, where <*id*> is the job ID. A subjob can be specified in the form *j*<*job-id*>.<*subjob-id*>, or by its process ID.

By default, *jobs* will display the current status of all jobs. The jobs upon which it operates, and the operation that it performs, can be affected by options and arguments; if multiple options are used, job and operation selections are ORed. If job or subjob IDs are given as arguments, they are selected, overriding the default.

When the status of a job or subjob is displayed, the fields displayed are, in order:

- The ID number, in brackets.
- The status, as a single character in the style of ps (R = runnable, S = sleeping/blocked, T = suspended, Z = finished). (for subjobs only)
- The process ID, in parentheses.
- The command that the job or subjob is running.

UNIX equivalent	jobs
DOS equivalent	None Applicable

Options

-A	Select all jobs. This is the default selection, but can be explicitly specified in order to combine it with other selections.
-C	Select all jobs that have completed running.
-d	Delete the selected jobs from the job table. If a job has completed running, this should be done at some point in order to avoid the job hanging around. If a job has not completed, it can be deleted anyway, and will continue running, but the shell will take no further interest in it; this is known as `disowning' the job. This option has no effect on subjobs, which are automatically deleted when they complete.
-l <variable>	Store the IDs of selected jobs and subjobs in the specified shell variable. Each ID is stored in a separate element of the variable.
-p	Print (to standard output) the status of the selected jobs and subjobs. This is the default action, but can be explicitly specified in order to combine it with other actions.
-R	Select all the jobs that have not completed running.
-s	If displaying the status of a job, also display all of its subjobs. This affects only the display; the subjobs are not considered to be selected.

10. Commands K and L

10.1 Kill -

Synopsis

```
kill -s SIGNAL pid ...
```

Description

The default signal is SIGTERM. Use '-l' or '-L' to list the available signals. Signals may be specified in three ways, -9, -SIGKILL, -KILL.

Examples

```
  kill 123 456                          Send SIGTERM to pids 123 and 456
  kill -s HUP 123                       Send SIGHUP to pid 123
  kill -s SIGHUP 123                    Send SIGHUP to pid 123
  kill -SIGHUP 123                      Send SIGHUP to pid 123
  kill -HUP 123                         Send SIGHUP to pid 123
  kill -1 123                           Send SIGHUP to pid 123
  kill -l                               List all signal names supported
  kill -L                               List signals in a table form
```

10.2 *ktrace* - Copy Arguments to Kernel Trace Stream Log File.

Synopsis

```
 ktrace [<arg> ...]
```

Description

Copies its arguments in turn to the kernel trace stream, separated by spaces and then followed by a newline.

UNIX equivalent	logger
DOS equivalent	None Applicable

Options

-n	Suppress the final newline.

10.3 *local* - Execute Specified Command and Restore Named Variable to Original Status

Synopsis

```
 local [<var> ...] <command>
```

Description

Executes the specified command (the exit status is the exit status of that command). After executing the command, the state of the named variables is restored to what it was prior to executing the command.

UNIX equivalent	`local var;`
DOS equivalent	None applicable

10.4 *ls* - List Files

Synopsis

```
ls [<filename> ...]
```

Description

Lists the named files. If a directory is named, its contents are listed. If no filenames are given, the contents of the current directory are listed instead.

UNIX Equivalent	ls
DOS Equivalent	dir

Options

-1	Output filenames one per line. This reverses the -C, -l, -m and -x options.
-A	Show all files in a directory except for "." and "..".
-a	Show all files in a directory. By default files whose names begin with "." are hidden.
-B	When expanding a directory, do not show files whose names end in "~".
-b	Quote non-graphic characters in filenames in a C-like manner. Reverses -N, -Q and -q.
-C	Display filenames in vertical columns. This is the default state, and reverses the -1, -l, -m and -x options.
-c	If displaying or sorting by timestamp, use the inode change time instead of the modification time.
-d	If directories are named on the command line, display entres for them, instead of listing their contents.
-e	In a long listing (-l), do not abbreviate the timestamp
-F	Add a character after the filename indicating the file's type. / directory @ symbolic link | named pipe % character special device # block special device * regular file that is executable
-G	Do not display group ownership. This is the default state, and the opposite of the -g option.
-g	In a long listing (-l), also display the file's group, after its owner.
-i	Display the inode number of each file.
-I <pattern>	When expanding a directory, do not show files whose names match the specified *glob* pattern.
-L	When processing a symbolic link, display the information for the file it points to, rather than for the link itself.
-l	Display files one per line, giving extra details. This reverses the -1, -C, -m and -x options. From left to right the data listed by -l is, the file's mode, link count, owner, size, timestamp, and name. For symbolic links, the target of the link is also listed. The file mode is listed as a sequence of ten characters. The first character gives the file type: • d directory • l symbolic link • p named pipe • c character special device • b block special device • - regular file The timestamp (by default the modification timestamp) is listed as month, day and year. Timestamps within the past six months or next hour have the time of day listed in place of the year.
-m	Display filenames across the screen, separated by commas, fitting as many per line as possible. This reverses the -1, -C, -l and -x options.
-N	Display filenames literally, including unprintable characters. Reverses -Q, -b and -q.
-n	In a long format listing, display the owner and group as numeric UID and GID respectively. By default, names are displayed if possible.
-p	Same as the -F option.
-Q	Quote non-graphic characters in filenames in a C-like manner, as -b does, but also enclose

	the entire filename in double quotes if it contains any non-graphic characters. Reverses -N, -b and -q.
-q	Display unprintable characters in filenames as "?". This is the default. Reverses -N, -Q and -b.
-R	Recursively list directories encountered.
-r	Reverse the sort order.
-S	Sort files by size, largest first.
-s	Display the size of each file, in 1K blocks.
-t	Sort files by timestamp (by default the modification timestamp), most recent first, instead of alphabetically.
-U	Sort files in the order in which they appear in the directory.
-u	If displaying or sorting by timestamp, use the last access time instead of the modification time.
-w <columns>	If displaying filenames in columns, act as if the screen width is that specified. This defaults to 80.
-X	Sort files alphabetically by extension (the part of the filename comprising the last "." and all characters after it). Filenames with no extension sort before all others. Files with identical extensions are sort ed alphabetically by full name.
-x	Display filenames in columns, sorted horizontally. This reverses the -1, -C, -l and -m options.

11. Commands M

11.1 Memtest - Performs memory testing.

Synopsis

Memtest was created to perform memory tests on embedded platforms, with options for soak testing.

Description

The minimum parameter requirements are to define the memory size (using the -m option), in which case a block of that size will be allocated from the free memory pool and tested. Alternatively, an absolute block of memory may be tested either by specifying a start address (using the -s option) and a size (using the -m option; optional, defaulting to 8k), or by specifying a start address (using the -s option) and an end address (using the -e option).

Tests performed on the memory block include:

- Fill test with XX: All words of the block are filled with that value, and then read back.
- Poke each word with its own address: Each 32 bit word has its 32 bit pointer value written to it. This checks for duplicate memory regions and "wrap around" within the block under test. eg "stuck" bits within the address bus.
- Pause (2 seconds): The memory block is not accessed during this time. If the memory refreshing circuitry is not working properly, the data will "fade", and will not verify correctly.
- Verify each word contains its own address:
- Poke each word with pseudo random value:
- Verify ...: Verification that each word within the block contains the expected value.
- Walking ones test: Check for "stuck bits". Bits may be stuck to one or zero within the data bus.

Note that testing absolute memory blocks may cause memory exceptions. The test count is displayed at the start of each test. This is useful when using the -l option, to test multiple times. The test count counts upwards.
Options

-s <*start_address*>	Specify the start address of the area to test. <start_address> is a number, in hexadecimal if prefixed by 0x, and multiplied by 1024 if suffixed by k (kilo) or by 1024*1024 if suffixed by M (Mega). The address must be word aligned.
-e <*end_address*>	Specify the end address of the area to test. <end_address> is a number, in hexadecimal if prefixed by 0x, and multiplied by 1024 if suffixed by k (kilo) or by 1024*1024 if suffixed by M (Mega). The address must be word aligned. The -e and -m options cannot be used together.
-m <*memory_size*>	Specify the size of the area to test, defaulting to 8k. <size> is a number, in hexadecimal if prefixed by 0x, and multiplied by 1024 if suffixed by k (kilo) or by 1024*1024 if suffixed by M (Mega). The size must be a multiple of 4. The -e and -m options cannot be used together.
-l <*loop_count*>	Specify the loop count, defaulting to 1. 0 means loop forever. <loop_count> is a number, in hexadecimal if prefixed by 0x.
-q	Perform quick tests (skip walking ones tests).

Example 1

Specify a *start_address* plus *size* to test an absolute block of memory. Test the 65536 bytes of absolute memory starting at byte 1048576 (0x00100000).

```
memtest -s 1M -m 64k
```

Example 2

Test a block from the free memory pool. The block is returned to the free pool after testing.
memtest -m 64k. Output from a test run on Win32:

```
Start of memtest
```

```
Using malloc
Start Address = $E1ED78
End Address   = $E2ED78
Size          = 65536 bytes
Loop count    = 1

Top 1
  Fill test with $0: OK
  Fill test with $FFFFFFFF: OK
  Fill test with $AAAAAAAA: OK
  Fill test with $55555555: OK
  Poke each word with its own address:  OK
  Pause (2 seconds): OK
  Verify each word contains its own address:  OK
  Poke each word with pseudo random value:  OK
  Pause (2 seconds): OK
  Verify each word contains random number written:  OK
  Walking ones test: lsl OK
  Walking ones test: lsr OK
  Walking ones test: asl OK
  Walking ones test: asr OK

Loop count reached zero.  No errors
```

11.2 *mkdir* - Create New Directories

Synopsis

```
mkdir <pathname> ...
```

Description

Creates directories with the pathnames that have been specified.

UNIX equivalent	`mkdir`
DOS equivalent	`md`

Options

-m <mode>	Specifies the permissions to apply to the directory created. This may be specified either in octal or as a symbolic mode, using the syntax of *chmod*.
-p	Also create all necessary parents in order to create each directory. Do not complain about directories that already exist.

11.3 *mv* - Rename Files

Synopsis

```
mv <source> ... <directory>
mv <source> <destination>
```

Description

This function renames the files that have been specified by the source arguments. If the last argument happens to be a directory, all the source files will be renamed to names which are within the target directory and have the same basename as the source argument. Otherwise there must be exactly two arguments, and the first one is renamed to the second. By default, *mv* will query the user before replacing any unwritable files.

UNIX equivalent	`mv`

DOS equivalent	`move`

Options

-f	Never query the user.
-i	Query the user before replacing any file.

12. Commands N and O

12.1 *not* - Execute Specified Command

Synopsis

```
not <command>
```

Description

This function executes the specified command. The exit status is zero if the command returned a non-zero status, and 1 otherwise.

UNIX equivalent	!
DOS equivalent	None applicable

13. objdump

Print information about object files.

Synopsis

```
objdump [<options>] <filename> [<filename> ...]
```

Description

Objdump prints information about object files of supported formats. Currently, the only supported formats are int**e**nt symbol files and int**e**nt tools. objdump is a standard program on most Unix systems, and the int**e**nt implementation recognises the same options (though some are unsupported).

Principal Options

-b <bfdname>	
--target=<bfdname>	Specify the target file type (overrides auto-detection). Valid names can be determined using the "--info" option.
--file-headers	Display information from the file headers
--section-headers	Display information from the section headers, if any

-i	
--info	Print the list of recognised machine types and target types.

-j <section>	
--section=<section>	Display information about the named section only.

-k	
--raw	Display information in raw mode (hex dump).

-m <machine>	
--architecture=<machine>	Specify the machine type (overrides auto-detection). Valid names can be determined using the "--info" option.

-q	
--quiet	Do not print extraneous information.

-s	
--full-contents	Print the contents of selected sections.

-t	

--syms	Print the symbols in selected sections.

-x	
--all-headers	Equivalent to specifying --file-headers, --section-headers and --syms.
--version	Display the version number of objdump.
--help	Display usage information for objdump.

13.1 *od* - Dump Files in Octal and Other Formats

Synopsis

```
od [<filename> ...]
```

Description

Concatenates all the named files, or reads standard input if none are specified. The resulting input is displayed on standard output in a configurable format.

UNIX® equivalent	od
DOS® equivalent	None Applicable

Options

-A <address-base>	Sets the format in which offsets into the input will be displayed. "d", "o" and "x" specify decimal, octal and hexadecimal, respectively. "n" suppresses the display of offsets. The default is octal.
-j <skip>	Before starting to generate output, skip the specified number of bytes of input. The amount to skip may be specified in decimal, octal with a leading "0", or hexadecimal with a leading "0x". A multiplier may be appended: "b" for POSIX blocks (512 bytes), "k" for kilobytes (1024 bytes), or "m" for megabytes (1048576 bytes).
-N <count>	Process no more than the specified number of bytes of input. The number may be specified in decimal, octal with a leading "0", or hexadecimal with a leading "0x". A multiplier may be appended: "b" for POSIX blocks (512 bytes), "k" for kilobytes (1024 bytes), or "m" for megabytes (1048576 bytes).
-t <format>:	Format data in the specified format. Each input block will be treated as chunks of data of the specified type, and displayed accordingly. Multiple formats can be concatenated, in which case each input block will result in multiple output lines, one per format. The formats that can be specified are: • a named ASCII characters • c characters • d signed decimal integers • u unsigned decimal integers • unsigned octal integers • x unsigned hexadecimal integers • f floating point Formats "a" and "c" take input in chunks of single bytes. Formats "d", "u", "o" and "x" by default take input in chunks of four bytes. They can be appended by a decimal number, giving a chunk size, or by a letter "C" (1 byte), "S" (2 bytes), "I" (4 bytes) or "L" (8 bytes). Format "f" by default takes input in chunks of 4 bytes (single precision). It can be followed by a letter "F" (single precision, 4 bytes), "D" (double precision, 8 bytes) or "L" (long double, 8 bytes).
-v	Display all relevant input, without abbreviation. By default, any number of output lines that are identical to the previous output line will be deleted, and replaced by a line containing only "*".

13.2 *or* - Execute Second Command in Event of First Failing

Synopsis

```
or <command> ...
```

Description

This function executes the first command that has been specified. If that exits with a non-zero status (indicating failure), the second command (if any) will be executed. It will then continue until either all commands have been executed, or one of them exited with a zero status, which is the exit status of the last command that has been executed.

UNIX equivalent	`||` for instance, `{...cmd1...}||{...cmd2...}`
DOS equivalent	None applicable

14. Commands P

14.1 *parse* - Parse Specified String

Synopsis

```
parse <string> [ {<var> <string>} ... ] [<var>]
```

Description

The specified strings are concatenated, and an attempt made to parse the result. For each variable name specified, the parser state stack at that point in the string is saved into the named variable. The variable's value contains one character per stack element, with the top of the stack at the left. The characters defined are:

- a: argument (ARG)
- c: command (CMD)
- l: command list (CMD-LIST)
- n: fd number (DIGIT-seq)
- p: pipe redirection (PIPE-REDIR)
- P: pipe specifications (PIPE-SPECS)
- r: simple redirection (REDIR)
- R: command redirection (CMD-REDIR)
- s: simple command (SIMPLE-CMD)
- v: unquoted variable name (EXPANSION)
- V: quoted variable name (EXPANSION)
- w: linear whitespace (LWSP)
- W: whitespace (WSP)
- x: ${} expansion (EXPANSION)
- X: expansion modifier (MODIFIER)

See the shell grammar for more information. Note that the stack saved refers to the parser state at the end of the preceding string; this does not necessarily indicate the role played by the next character.

If a parse error occurs, then all remaining unprocessed variables are set to a string consisting of a `!' followed by the character at which the parse error was detected.

Options

-e <var>	Treat the specified variable like the variables named in command line arguments, except that it is considered to be placed after the end of input token. After a successful parse, therefore, the variable is set to the empty string. On a parse error it is set as described above; if the parse error occurs at the end of the string (which is not otherwise detectable), the variable is set to the string "!".
-w	For each variable set indicating the parser stack, store the canonical textual form of the words of the innermost enclosing simple command (`s' state) in the second and subsequent elements of the variable, one word per element.

UNIX equivalent	None Applicable
DOS equivalent	None Applicable

14.2 *paste* – Paste Lines From Specified Files

Synopsis

```
paste <filename> ...
```

Description

Reads lines from the specified files, sending to standard output lines that are the concatenation of corresponding lines from the files, in the specified order. By default, the lines being concatenated are separated by tab characters.

Options

-d <characters>	Uses the specified characters, in order, repeating circularly, to separate lines being concatenated. The character list is reread from the beginning for each output line. The following sequences have special meaning: • "\n": newline character • "\t": tab character • "\\": "\" character • "\0": no separator (not a NUL character)
-s	Concatenate all the lines in each file in turn, so that each input file generates one line of output.

14.3 *printf* - Display Arguments

Synopsis

```
printf <format> [<arg> ...]
```

Description

Displays the optional arguments, formatted in accordance with the format argument, in a manner reminiscent of (though not identical to) *lib/printf*.

Most characters in the format argument are copied to standard output unmodified. All C backslash escapes are understood. Format specifiers begin with "%"; each format specifier determines how the next available argument is displayed. The format string will be processed as many times as necessary to use up all the arguments. Format specifiers for which there is no corresponding argument format a zero value or empty string.

As an exception to the general rule, "%%" is not a format specifier, but rather an escape resulting in a literal "%" character.

The "%" of a format specifier is followed by, in order:

- Zero or more flags. The interpretation of flag characters "+", " ", "0" and "#" depends on the conversion type. "-" causes padding to be added at the right instead of the left.
- Optional field width in decimal. This defaults to zero. This is the minimum field width; if there are more characters to display, the field will expand as necessary. If there are insufficient characters to fill the field, it will be padded, by default with spaces on the left.
- Optional precision. This consists of a decimal point (".") followed by zero or more decimal digits. The interpretation, and default value, depend on the conversion type.
- Conversion character (see below).

Conversion Characters

u	The argument is an integer value, specified as an optional sign followed by one or more decimal digits. A leading "0" means that the digits are interpreted in octal, and a leading "0x" or "0X" makes it hexadecimal. If the first character of the argument is a quote character ("'" or "'"), then the value of the second character is used. The integer value is displayed in decimal. The precision defaults to 1, and specifies the minimum number of digits to generate. If the value is negative, a leading "-" sign is added. If the "0" flag is used, and no precision is specified, then padding on the left is performed by adding leading "0" characters before the digits.
d, i	As "u", except for the interpretation of flags. " " causes a leading space to be added to non-negative numbers. The flag "+" causes a "+" sign to be added to non-negative numbers. Of these

	two, "+" takes precedence.
o	As "u", except that the value is displayed in octal. The "#" flag increases the precision such that the first digit is a "0".
x	As "u", except that the value is displayed in lowercase hexadecimal. The "#" flag causes "0x" to be added before the digits.
X	As "u", except that the value is displayed in uppercase hexadecimal. The "#" flag causes "0X" to be added before the digits.
c	A single character, the first character of the string argument, is displayed.
s	The string argument is displayed. The precision specifies the maximum number of characters to display, defaulting to all of them.
b	As "s", except that certain backslash escapes are recognised in the argument. The normal C backslash escapes are accepted, except that octal escapes may have up to four digits. "\c" terminates the argument, and also stops the processing of the format string.

UNIX® equivalent	`printf`
DOS® equivalent	None Applicable

14.4 Prng- Generates a pseudorandom bit sequence.

Synopsis

```
prng <algorithm> [<seed>]
```

Description

Generates a pseudorandom bit sequence, which is sent to standard output. The <algorithm> specified must be a class derived from lib/prng. By default, "lib/prng/" is prepended, so, for example, the RC4 keystream generator, implemented in the class lib/prng/rc4, can be specified as "rc4". If the name specified starts with a "/", it is taken as an absolute class name. In any case, "/_new" will be appended to get the name of the allocator tool.

If the <algorithm> string contains "//", then the rest of the string is taken as an option string, and is passed to the selected PRNG algorithm class. The interpretation varies from class to class; for example, to the lagged Fibbonacci PRNG class it specifies the word size and tap positions. This argument defaults to an empty string, the interpretation of which is algorithm-dependent.

The <seed>, if specified, must consist of an even number of hexadecimal digits. This will be converted into a byte string, which will be passed to the PRNG as a seed value. Any length of seed may be specified; the exact interpretation depends on the PRNG algorithm. If no seed is specified, the current system time is used as the seed. prng does not terminate normally. If successful, it will eventually receive a SIGPIPE (when the output stream is closed), upon which it exits with status zero.

14.5 Prof- Displays information about memory objects in Elate system.

Synopsis

```
prof
```

Description
Prints information about each memory object in the running Elate system. The information printed for each memory object is as follows:

Chip#	The Elate processor number on which the memory object resides.
Memory	The amount of available memory in the memory object
Maxblock	The size of the largest available memory block in the memory object
#Frags	The number of fragments into which the available memory in the memory object is split

Total	The total amount of memory in use by the memory object (allocated+free)
Memobj	The symbolic name of the memory object The class tool of the memory object

Exit Status

prof will exit with a status of 0 (indicating success) if all information was displayed successfully. If there was an error retrieving any of the required information, or if invalid parameters are specified, prof exits with a status of -1.

14.6 *ps* - Display Process States

Synopsis

```
ps [<pid-list> ...]
```

Description

This function displays the status of processes. The requisite processes may be specified by their process ID, but otherwise the status of all processes will be displayed.

UNIX equivalent	ps
DOS equivalent	None Applicable

Options

-h	Do not output a header line.
-o <column-list>	Overrides the default output format. <*column-list*> is a space- or comma-separated list of the names of columns to display. Multiple -o options may be given; their lists are concatenated. The default header for a column can be overridden by following a name with an `=`; everything after the `=` up to the end of the argument is used as the column header. If all column headers are blank, the header line will be suppressed.
-p <pid-list>	<*pid-list*> is a space- or comma-separated list of process IDs. Display the status of those processes. (Non-option arguments are treated in exactly the same way.)

Column types

The -o option can be used to select the following column types:

args	The process' argument block. This is initially the process' command line arguments, but can be modified.
comm	The main tool of the process.
pid	The process ID.
ppid	The process's parent's PID.
state	A character indicating what the process is doing: • R Runnable (either running or ready to run) • S Sleeping (voluntarily) • T Suspended • Z Dormant (waiting to be started or reaped) • - Dead (being deleted)

14.7 *pwd* - Display Current Directory Name

Synopsis

```
pwd
```

Description

Displays the name of the current directory (the value of the environment variable *lib/absname*) to standard output.

UNIX equivalent	`pwd`
DOS equivalent	`cd`

15. Commands R

15.1 *read* - Read From a File Descriptor

Synopsis

```
read <variable>
```

Description

This function will read a line of text from standard input, until it reaches a newline character. It will then strip off the newline and place the result in the specified variable. If end of file is read, the data read up to end of line is placed in the variable, and the return status is 1 instead of 0.

If an error occurs in reading, an exception *read.err* will be thrown.

UNIX equivalent	`read`
DOS equivalent	None Applicable

Options

-c	Read only one character, and do not treat newline specially.
-m	Read multibyte characters, instead of single byte characters.

15.2 *return* - Return From Function With Specified Status

Synopsis

```
return [<status>]
```

Description

Returns from the current function, with the specified status (default zero). This is actually accomplished by throwing an exception, which is then caught by the function execution code.

UNIX equivalent	return
DOS equivalent	None applicable

15.3 *rm* - Remove Files

Synopsis

```
rm <pathname> ...
```

Description

This function removes the files specified by the arguments. By default, directories cannot be removed, and *rm* will query the user before removing any unwritable files.

UNIX equivalent	`rm`
DOS equivalent	`del`

Options

-d	Attempt to remove directories as if they were regular files.
-f	Never query the user. Also do not display an error message, or change the exit status, when an attempt is made to remove a file that doesn't exist.
-i	Query the user before removing each file.
-r	Recursively remove directories: the files in each directory are processed as if they were specified as

	command line arguments, then the directory itself is removed in the manner of rmdir.
-R	A POSIX-mandated synonym for -r.

15.4 *rmdir* - Remove Directories

Synopsis

```
rmdir <pathname> ...
```

Description

Removes the directories specified by the arguments. Non-directory files and non-empty directories cannot be removed.

UNIX equivalent	`rmdir`
DOS equivalent	`rmdir / rd`

Options

-p	After removing the full pathname specified by an argument, drop its last component, and process the remaining prefix similarly. For example, *rmdir -p a/b/c* has much the same effect as *rmdir a/b/c a/b a*.

15.5 Rpt - Runs a VP command every x seconds.

Synopsis

```
rpt <time> <target> [<arg> ...]
```

Parameters

<time>	is an integer giving the number of seconds to wait before repeating the command.
<target>	is the pathname of the command to spawn.
[<arg> ...]	are optional arguments to be passed to the spawned command.

Description

Spawns a process to run the supplied command (with any parameters). Waits for the required number of seconds, and respawns the process. Note that if the previous invocation has not already exited, 'rpt' exits with an error.

15.6 *run* - Execute Command via Path Search

Synopsis

```
run <command> [ <arg> ... ]
```

Description

Execute the specified <*command*>, via a shell-style path search. The command is given the specified arguments, with the command name being the first argument. Note that if any of the arguments to pass on could start with "-", for example if there are options, then a "--" argument must be used to avoid such arguments being interpreted as options to the run command.

The exit status is the exit status of the specified command, or 1 if an error occurred.

UNIX equivalent	None applicable
DOS equivalent	None applicable

Options

-0	Do not pass on the command name as the first argument. This allows a command to be

	run under a different name. If no arguments are provided after the command name, the command will get no arguments at all.
-b	Do not wait for the command to finish before exiting. This option implies -s, and disables -t.
-p <num>	Run the command on the specified processor. The default is the local processor.
-s	If successful, exit with a status of zero, rather than the command's exit status.
-t	When the command has completed, output timing statistics to standard error.

16. Commands S

16.1 *sed* – The Stream Editor

Synopsis

```
sed [<script>] [<filename> ...]
```

Description

sed is the stream editor. It concatenates the specified input files (or standard input if none are specified), then processes this input in accordance with the specified script. The script can be specified as either the first command-line argument, or by any combination of -e and -f options, but not both.

In the main processing loop, first a line is read from the input into the pattern space. Then the commands of the script are executed. If control reaches the end of the script, the (possibly modified) contents of the pattern space are written to standard output, unless this was disabled (with the -n option). Then the pattern space is emptied, and control goes back to the start of the loop - the next line of input is read.

Normally commands are processed in sequence. The b, t and : commands can be used to perform explicit branches. The c, d, D and q commands also perform implicit branches.

Some commands make use of a hold space. This is a buffer much like the pattern space, but its contents can only be affected explicitly. It is initially empty.

When a line has been read into the pattern space, the newline is not included. A newline is added again when the pattern space is output. When there are multiple lines in the pattern space, they are separated by newlines.

UNIX® equivalent	sed
DOS® equivalent	None Applicable

Regular Expressions

sed regular expressions are POSIX Basic Regular Expressions, except that the sequence "\n" can be used to represent a newline. Regular expressions are always delimited by a specific character, most commonly "/", though any character except backslash or newline can be used.

Within the regular expression, the delimiter and newline can each be escaped by a precedingbackslash; unescaped newlines are not permitted. However, if the delimiter is a metacharacter when preceded by backslash, (e.g. "("), then the special meaning will be applied. Also, it is impossible to escape any character in a bracket expression. Consequently, it is unwise to use as a delimiter any character that needs to be used anywhere in the pattern.

Addresses

Each command in a sed script can be preceded by zero, one or two addresses. If two addresses are used, they are separated by a comma. A command with no address specified is always applied. A command with one address is applied whenever that address matches. A command with two addresses is applied first when the first address matches, and then always until the second address matches, after which it starts testing the first address again.

The following address forms are supported:

<number>	A number matches if it is the current line number. (The current line number is the number of lines read from the input so far.) Exception: if used as the second of a pair of addresses, it also matches if the current line is a later one than that specified. Thus the command "3,1p" will be applied only on one line (line 3).
$	Matches if this is the last line (i.e., all the input has been read).

/<BRE>/	Matches if the contents of the pattern space matches the specified regular expression, delimited by "/".
\#<BRE>#	Matches if the contents of the pattern space matches the specified regular expression, which can be delimited by any character except for backslash or newline ("#" is used in this example).

Commands

Commands are separated by newlines or semicolons, and preceding and trailing whitespace is permitted. Exception: comments (#), commands taking filename arguments (r and w), and commands taking text arguments (a, c and i), can be terminated only by a newline.

{ <command> ... }	Groups commands. Only useful if preceded by an address.
a\<text>	Write the specified text to standard output, just before the next input line is read (by the n or N commands, or the implicit read at the start of the cycle) or before quitting with the q command. The text must consist of one or more complete lines, terminating at the first unescaped newline. Any character can be escaped by preceding it with a backslash.
b [<label>]	Jump to the specified label, or to the end of the script if no label is specified.
c\ <text>	Write the specified text to standard output, except that if the command has a double address, only perform this output on the last line of the matching range. Then delete the pattern space and start the next cycle, in the manner of the d command. The text must consist of one or more complete lines, terminating at the first unescaped newline. Any character can be escaped by preceding it with a backslash.
d	Delete the pattern space, and start the next cycle (by reading the next line of input and then branching to the top of the script).
D	If the pattern space contains a newline, delete the part of the pattern space up to and including the first newline, and branch to the top of the script. Otherwise behave like d. (The difference between these two cases is that the next line of input is read if and only if there was only one line in the pattern space to delete.)
g	Copy the contents of the hold space into the pattern space.
G	Append the contents of the hold space to the pattern space, inserting a newline separator.
h	Copy the contents of the pattern space into the hold space.
H	Append the contents of the pattern space to the hold space, inserting a newline separator.
i\ <text>	Write the specified text to standard output. The text must consist of one or more complete lines, terminating at the first unescaped newline. Any character can be escaped by preceding it with a backslash.
l	Write the contents of the pattern space to standard output, in a visually unambiguous form. Newlines are written as a "$" followed by newline; long lines are split with a backslash newline sequence; and backslash and non-printable characters are written as C backslash sequences.
n	Write the contents of the pattern space to standard output, unless this was disabled (with the -n option). Then read the next line of input into the pattern space. This is what happens implicitly if control reaches the bottom of the script, except that this command does not then branch to the top of the script.
N	Append the next line of input to the pattern space, separating it from the current contents with a newline. Note that the current line number changes, and that sed will terminate if there is no more input.
p	Write the contents of the pattern space to standard output, terminated by a newline.
P	Write the initial part of the pattern space, up to the first newline (or the whole pattern space if there is no newline), to standard output, terminated by a newline.
q	Write the contents of the pattern space to standard output, unless this was disabled (with the -n option). Then terminate.
r <filename>	Write the contents of the specified file to standard output, just before the next input line is read (by the n or N commands, or the implicit read at the start of the cycle) or before quitting with the q command. If the specified file can't be opened, do nothing.
s/<BRE>/<string>/<flag	Substitute the specified <*string*> for the first match of the specified regular

s>	expression in the pattern space. (Does nothing if there is no match.) Each instance of " & " in the replacement string is replaced by the entire matching portion of the pattern space, and "\1", "\2", etc. are replaced by the corresponding matching subexpression. Any character other than digits can be escaped in the replacement string by preceding it with a backslash. The delimiter ("/" here) can be any character except for backslash or newline. The flags can be: • g - Substitute all substrings matching the pattern, not just the first. • <number> - Substitute the <*number*>th matching substring, instead of the first. • p - If a substitution is made, write the contents of the pattern space to standard output, in the manner of the *p* command. • w <filename> - If a substitution is made, write the contents of the pattern space to the specified file, in the manner of the *w* command.
t [<label>]	Jump to the specified label, or to the end of the script if no label is specified, if any substitutions have been performed since the last line was read or the last t command executed.
w <filename>	Write the contents of the pattern space to the specified file, terminated by a newline. The filename may be separated from the w by whitespace.
x	Exchange the contents of the pattern and hold spaces.
y/<string1>/<string2>/	Replace all instances of characters in <string1> with the corresponding characters in <string2>, in the manner of the tr utility. The delimiter ("/" here) can be any character except for backslash or newline. Any character can be escaped in either string by preceding it with a backslash.
! <command>	Execute the specified command if and only if the address is not matched. If no address is used, or more than one consecutive ! is used, the net effect is a no-op.
: <label>	Do nothing. This command defines a label that can be used as the target of a b or t command.
=	Write the current line number to standard output.
[#<text>]	An empty command is ignored. A commant, which starts with "#" and continues to the end of the line, is ignored. Exception: if the first two characters of the script are "#n", it is equivalent to using the -n option.

Options

-e <commands>	Append the specified commands to the editing script.
-f <filename>	Append the commands contained in the specified file to the editing script.
-n	Omit the implicit output command from the end of the script. The same effect is obtained if the first two characters of the script are "#n".

16.2 *set* - Set Variable

Synopsis

```
set <var> [ <arg> ... ]
```

Description

Sets the value of the specified shell variable to the array consisting of the arguments which have been specified.

UNIX equivalent	`var=value`
DOS equivalent	`set var=value`

16.3 *shutdown* - Leave intent

Synopsis

```
shutdown [ <status> ]
```

Description

Shuts down the intent system.

If specified, *status* will be the exit status returned from intent to the host OS (if any). If *status* is not specified, the exit status will be zero.

It is strongly recommended that this function should be used to exit intent, rather than by simply deactivating the intent session.

UNIX equivalent	`Shutdown`
DOS equivalent	None applicable

Options

-n	Shut down immediately, without syncing. By default, *lib/sync* will be called to sync all data to disk, before shutting down.

16.4 *sleep* - Send Process to Sleep for Specified Time Length

Synopsis

```
sleep <time>
```

Description

Sleeps for the number of seconds specified by the argument, then terminates. As an extension to the standard POSIX functionality, the <*time*> argument may have a decimal fraction part. The time specified is the minimum delay length. The delay may be arbitrarily longer due to timer resolution or scheduling.

UNIX® equivalent	sleep
DOS® equivalent	None Applicable

16.5 *sort* - Sorts Lines In Files.

Synopsis

```
sort [<file> ...]
```

Description

Sorts the lines in the input files (default standard input), sending the sorted lines to standard output. Options specify the type of sorting to perform.

Options

-b	Ignore leading blanks when locating sort keys.
-d	`Directory' sorting: consider only blanks and alphanumerics when comparing.
-f	Fold lines to uppercase when comparing.
-i	Ignore non-printable characters when comparing.
-k <keydef>	Defines a sort key. Multiple sort keys may be given; they are significant in the listed order. If no -k options are given, a default of "-k1". The key definition consists of a start point specification and an end character specification, separated by a comma. The end character specification and preceding comma are optional; if omitted, the key continues to the end of the line. Each part consists of a field number, optionally followed by a "." and a character number within the field. If the character number is omitted, it defaults to the first or last character of the field, for the start and end specification respectively. Either the start or end specification may be followed by zero or more of the letters "b", "d", "f", "i", "n" or "r". Each such letter applies the effect of the corresponding option to the sort key. "b" affects only the part to which it is attached, but the others affect the interpretation of the entire key. If no options are attached to the key definition, then those options that have been specified separately, preceding the key definition, are applied.

-n	In each sort key, skip leading blanks, then examine the following numeric string (which may include a leading minus sign), and compare numerically. If there are no digits, the field compares as a value of zero.
-r	Reverse the sort order.
-s	Stable sort: lines that compare as equal based on key comparisons shall be output in the order in which they appeared in the input. -r has no effect on this. (By default, lines that compare equal are, as a last resort, compared byte-by-byte, respecting -r.)
-t <char>	Specifies the field separator. Each line is divided into fields separated by the specified character; by default the field separator is any sequence of blanks following a non-blank. These fields are used in key definitions (see -k).
-u	Only output the first of each set of lines that compare equal based on keys.

16.6 *source* - Execute Shell Command Read from Specified File

Synopsis

```
source <filename>
```

Description

Executes shell commands read from the specified file, within the current shell environment. Then source returns the exit status of the last command executed, or zero if there were no non-null commands.

This is what the shell does if invoked with a filename argument (to execute it as a script).

UNIX equivalent	"."
DOS equivalent	file.bat

16.7 *speed* - Benchmark Current Processor Speed

Synopsis

```
speed
```

Description

Benchmarks the speed of the current processor, and displays the results to standard output.

UNIX equivalent	None applicable
DOS equivalent	None applicable

16.8 *sr* - Write Self Reproducing Program to Standard Input

Synopsis

```
sr
```

Description

This function writes a self-reproducing program to standard output.

UNIX equivalent	None Applicable
DOS equivalent	None Applicable

16.9 *ssm* - Performs Miscellaneous Serial Port Operations.

Synopsis

```
ssm [ <command> <arguments> ]
```

Description

ssm is a client for development boards that run the SingleStep monitor. It allows arbitrary data to be transferred to and from the monitor, it has a built-in terminal, and it has a number of other assorted features that come in useful when dealing with development boards.

The program is divided into a number of subcommands. A list of the available commands can be obtained by typing "ssm help".

ssm uses the serial port to communicate with the development board, and needs to be set up before it can be used.

All settings are stored by *ssm* between invocations. The configuration file is called *$HOME/.ssmrc*, and should not be manually edited. The current configuration can be viewed at any time.

Endianism

Development boards may be either big- or little-endian. As from the point of view of VP, memory is always little-endian, it may be desirable to treat the memory as little endian even on big-endian platforms.

The *setendian* command switches on and off endian swapping of data as it is transferred to and from the board. The data is transferred from the board in quads and swapped inside *ssm*, so when big-endian mode is turned on, all transfers must be quad-aligned.

It may be useful to change endianism quite frequently when debugging big-endian boards in order to look at native data (in little-endian mode) and VP data (in big-endian mode).

16.10 *stat* - Produce File Statistics

Synopsis

```
stat <var> [ <filename> ... ]
```

Description

The specified files are examined, and their statistics stored in the specified shell variable, one file per element. Each element of the array has the form of a space-separated list of data; the data are, in order:

- a character giving the file type (the *ls* characters are used)
- the file permissions, as a four-digit octal number
- the device number of the device on which the file resides
- the file's inode number
- the number of links to the file
- the file owner's UID
- the file's GID
- the file's size in bytes
- the file's atime (last access time) as a time_t
- the file's mtime (last modification time) as a time_t
- the file's ctime (last inode change time) as a time_t

If an error occurs, then an appropriate *shell.err.stat* exception is thrown, and the variable is unaffected.

16.11 *status* - Exit With Specified Numerical Status

Synopsis

```
status <status>
```

Description

Exits with the specified numerical status, for example *status 3*.

UNIX equivalent	None Applicable

DOS equivalent	None applicable

16.12 *strings* - Extracts Printable Strings

Synopsis

```
strings [<file> ...]
```

Description

Extracts printable strings from the specified files (default standard input). Apparent printable strings are written to standard output, one per line. A printable string is defined as a sequence of printable ASCII characters, terminated by a newline or NUL character.

Options

-a	Ignored, for compatibility with Unix and POSIX.
-n <num>	Specifies the minimum length of string to print. Defaults to 4.
-t <address-base>	Sets the format in which offsets into the input will be displayed. "d", "o" and "x" specify decimal, octal and hexadecimal, respectively. "n" suppresses the display of offsets. The default is that offsets are not displayed.

16.13 *strip* - Strips Debug Information and/or Header Extensions from a Tool

Synopsis

```
strip <filename> ...
```

Description

For each *<filename>*, the *strip* utility reads the file of that name as a tool, strips debug information and/or header extensions from it, and writes the resulting tool back to the same file.

By default, *strip* strips all of the debug information, and none of the header extensions. Options are provided to alter this behaviour.

Options

-d <spec>	Specify which debug sections to remove. The strip utility builds a list of which debug sections to strip by starting with a list with all possible debug sections, and then scanning the *<spec>* string as a sequence of records. Each record contains a + character to add something to the list, or a - character to remove something to the list, then one of: a * character to add/remove all possible types a number (with $ or 0x prefix for hexadecimal) to add/remove that number type or the symbolic name for a debug section type (listed in intent Debug Structure) Each section of debug information in the tool is then stripped if its type appears in the finished list, or kept if it does not.
-h <spec>	Specify which header extensions to remove. The strip utility builds a list of which header extensions to strip by starting with an empty list, and then scanning the *<spec>* string as a sequence of records. Each record contains a + character to add something to the list, or a - character to remove something to the list, then one of: a * character to add/remove all possible types a number (with $ or 0x prefix for hexadecimal) to add/remove that number type or the symbolic name for a header extension type (listed in Header extensions) Each header extension in the tool is then stripped if its type appears in the finished list, or kept if it does not.

Examples

To strip all the debug information and none of the header extensions:

```
strip demo/mytool.00
```

To strip just the JavaTM debug information and the "embeddable" header extension:

```
strip -d{-*+__DEBUGSECTION_JAVA} -h{+TOOLHDREXT_EMBEDDABLE} demo/mytool.00
```

To strip all debug information except the JavaTM debug information, and all header extensions except "embeddable":

```
strip -d{-__DEBUGSECTION_JAVA} -h{+*-TOOLHDREXT_EMBEDDABLE} demo/mytool.00
```

16.14 *stty* - Reads or Sets Terminal Parameters.

Synopsis

```
stty [<command> ...]
```

Description

stty provides a shell interface to the reading and setting of terminal parameters. See *dev/tty/tcsetattr* for the API.

stty does not perform standard option processing. Instead, each argument is taken as a command from the list specified in the *stty* documentation. The *tty* to be affected must be connected to *stty*'s standard input. Normally, in an interactive session, this will be the case anyway, but in any case this can be effected by shell redirection.

For precise descriptions of the semantics of all the *tty* parameters that can be changed, see the *termios* structure.

Commands

This is a selection of basic commands and is not comprehensive.

-l	List *tty* parameters to standard output. This is the default action if no commands are specified. Only parameters that have some effect are listed. Flags are explicitly displayed only where they differ from the `sane' state.
-a	List the *tty* parameters to standard output in the format mandated by POSIX.
-g	List all the *tty* parameters in a machine-readable form. The output consists of a non-empty string of printable non-whitespace characters, followed by a newline. This output can be used as a parameter to a later invocation of *stty*, with or without the final newline, to restore the *tty* parameters to precisely the state at the time -g was used.

sane	Reset *tty* parameters to the default (reasonable) state. (POSIX.2 actually defines this command using the word "reasonable".) This is equivalent to the effects of the *dev/tty/cfmakesane* tool.
raw	Unset all flags relating to processing of input and output. This makes the *tty* completely transparent - input and output proceed as if directly to the underlying devices. This is equivalent to the effects of the *dev/tty/cfmakeraw* tool.

rows <number>	Set the number of rows in the terminal window. Note that depending upon the display device being used, this option may resize the window, return an error code, or only affect the values returned when the window size is queried.
cols <number>	Set the number of columns in the terminal window.
columns <number>	This is a synonym for *cols*.
size	Return the current terminal size, in the format "rows cols".

16.15 *sub* - Execute Command Within Subshell

Synopsis

```
sub <command>
```

Description

Executes the specified command in a subshell. This means that the command is run in a separate process from the current shell environment, and changes it makes to its environment - redirections and shell variables - will not affect the current shell. *sub* returns the exit status of the command executed.

Note that the command must be a single argument. If it is necessary to run a more complex command (for example, including spaces), then you will need to quote the command using the normal mechanism.

UNIX equivalent	Command executed inside brackets, for instance `(...command...)`
DOS equivalent	None Applicable

Options

-b	Execute the command in the background; that is, do not wait for the subprocess to terminate before returning. *sub*'s exit status is zero.

16.16 *sync* - Flush Data To Disk

Synopsis

```
sync
```

Description

This function flushes all data to disk.

UNIX equivalent	`sync`
DOS equivalent	None Applicable

17. Commands T

17.1 *tail* - Output The Last Part of Files

Synopsis

```
tail [<filename> ...]
```

Description

This will display only the last part of each specified file. If no files are specified, standard input is used instead. The "last part" of a file is by default the last ten lines. If more than one filename is specified, the name of each file is displayed in a banner preceding the extract of the file.

UNIX equivalent	`tail`
DOS equivalent	None Applicable

Options

<sign><number>	Equivalent to *-n<sign><number>*. This must be the first option on the command line.
-c [<sign>]<number> [<multiplier>]	Equivalent to -n, but the units are bytes rather than lines. The number may be suffixed by a multiplier character: "b" for 512-byte blocks, "k" for kilobytes, or "m" for megabytes.
-n [<sign>]<number>	If the sign is not present or is "-", display the specified number of lines from each file. If the sign is "+", display lines from the specified line number. (*tail -n+1* will therefore display the entire file.)
-q	Do not display filename banners.
-v	Display filename banners, even if only one file is being processed.

17.2 *tee* - Copy *Stdin* to *Stdout* and Named Files

Synopsis

```
tee [<filename> ...]
```

Description

Copies standard input to standard output, and also to each of the named files. The name of this command comes from the plumbing analogy of stream redirection, in that it creates a T-junction in a pipeline.

UNIX equivalent	`tee`
DOS equivalent	None Applicable

Options

-a	Append to the named files, rather than overwriting them.

17.3 *throw* - Throw Exception

Synopsis

```
throw <exception> [<status>]
```

Description

This throws the specified exception. The specified numerical status (default zero) is used for this.

UNIX equivalent	None Applicable
DOS equivalent	None Applicable

17.4 *tooldump* - Examine Tool List

Synopsis

```
tooldump
```

Description

Examines the tool list, and displays the address range, reference count and name of each tool.

UNIX® equivalent	None Applicable
DOS® equivalent	None Applicable

Options

-b	Display only toolnames

17.5 *touch* - Update Timestamps

Synopsis

```
touch <filename> ...
```

Description

Update the timestamps on the named files to the current time. All files will be given the same timestamp, regardless of how long *touch* takes to execute. Named files that do not exist will be created.

UNIX equivalent	`touch`
DOS equivalent	None Applicable

Options

`-a`	Update only the last access time.
`-c`	Do not create files that do not already exist, and do not complain about the situation.
`-m`	Update only the last modification time.
`-r <filename>`	Use the timestamps of the named file, instead of the current time.

17.6 *tr* - Copy and Modify data

Synopsis

```
tr <string1> [<string2> [<string3>]]
```

Description

Copies standard input to standard output, modifying the data as specified by the options and arguments. If no options are given, there must be two arguments, and translation is performed. For instance if a text file has been created like so:

```
cat > mytext.txt
```

If that file has the lower case values abcdz, then it is possible to convert then to upper case by using the *tr* command as shown here:

```
tr a-z A-Z < mytext.txt
```

Each argument specifies a sequence of byte values. When translation is performed, byte values specified by the first string are changed to the byte value specified in the same location in the second string. If these sequences differ in length, the trailing part of the longer is ignored for the purposes of translation.

UNIX equivalent	`tr`
DOS equivalent	None Applicable

Options

-c	<*string1*> is taken to represent exactly those byte values that it would not otherwise include, in ascending order.
-d	There must be one argument. Delete all the byte values specified by <*string1*>. Translation is not performed.
-s	In addition to what would otherwise be done, squeeze all the byte values specified by the last argument. Consecutive repetitions of these byte values are changed to single copies. If deletion is also being performed, a second argument must be provided - the first is used to control deletion, and the second controls squeezing. If translation is also being performed, a third argument may be provided; if it is not, then the second argument controls both translation and squeezing. This option may also be used on its own, with only one argument, in which case squeezing is performed instead of translation. Note that the three argument form is an extension to the standard POSIX behaviour.

Argument specification

The arguments may contain the following special forms:

\ooo	(1-3 octal digits) byte with the specified octal value
\a	\a, \b, \f, \n, \r, \t, \v as in C
\x	anything else backslashed stands for itself
c-c	bytes within the specified inclusive range
[c*n]	n copies of c; n may be in decimal, or octal with a leading 0
[c*]	(in string2 only) enough copies of c to match length of string1
[=c=]	equivalence class
[:class:]	character class

Any character that is not part of one of these special forms stands for itself.

17.7 *translate* - Run Translator

Synopsis

```
translate [ <filename> ... ]
```

Description

For each specified file, or standard input if no file is specified, a translator is run on the tool or Java™ class contained in the file. The translated tool is stored in a file with the name of the tool with an appropriate *.nn* suffix added. Directories are created as necessary to do this. The exit status is zero if all input files were translated and written out successfully.

UNIX equivalent	None Applicable
DOS equivalent	None Applicable

Options

-c	Disable checks on the format of VP input. By default, a check translator is run validating the input before the native translator is run. This only applies to the VP translators. The verifying Java translator does these type checks for Java classes.
-e	Enable printing of detail with error message. When an error occurs translating (or verifying) a Java class, this option causes the single line of the disassembly of the class which contains the erroneous offset to be printed. For access errors, this provides information about which class and member were not accessible.
-g	Enable generation of debug data, if the translator has the capability, even if there

	is none in the input file. This will not affect the actual code being generated. The data generated allows debugging across the translation being performed. By default, debug data is generated only if the input file already contains some debug data. This option cannot be used in conjunction with -s.
-n	Do not flush the generated tools from memory. By default, each tool output is flushed from the tool list, so that the newly generated version will be used immediately, if it is being saved in the appropriate point in the filesystem. If this option is not specified then a warning will be output if it is not possible to flush the tool.
-o <number>	Set the translator flags word. The number may be specified as decimal, octal if it starts with '0', or hexadecimal if it starts with '0x'. By default, a flags word of 0 is used. The flags word specified (or the default of 0) is subject to further modification before being used by the translator: • If the -g directive is specified, bit 0 is set. • If the -s directive is specified, bit 1 is set.
-r <directory>	Set tool root. The specified directory is used as the tool root; the default is currently. (the current directory) but will probably change soon to / (the intent root). Setting the tool root affects both where translators are loaded from, and where translated tools are saved to. If the tool root is anything other than /, then the translator is loaded from its .00 file and translated, even if a pretranslated version of the translator exists.
-s	Strip debug data. This option ensures that there is no debug data in the output file, even if there is some in the input file. Many Java compilers output debug data in a class file by default; this option is useful to ensure that this does not cause debug data to appear in the translated tools when it is not required. By default, debug data is generated if the input file already contains some debug data. This option cannot be used in conjunction with -g.
-t <translator>	Use the specified translator (by default the current system translator is used). If the translator name specified starts with `/', it is used literally as the tool name, with the `/' removed. Otherwise the directories *sys/java/tr* and *sys/tr* are searched for the translator. In either case the location to load the translator from is taken to be relative to the tool root specified by -r. Specifying a tool which is not a translator yields unpredictable behaviour.
-T <iterations>	Call the translator the specified number of times, and output the total time taken. This option is useful for timing a translator; an iteration count larger than 1 is required on a system whose timer does not have a high enough resolution to measure a single translation.
-v	Write to standard error a report of the translator's name and version number, and one line per translated tool giving statistics.

17.8 *true* - Does Nothing

Synopsis

```
true [<arg> ...]
```

Description

Does nothing, ignores any arguments, and exits with a status of 0 (which conventionally indicates success). True can be used as a place holder in shell scripts where a successful command is needed.

UNIX equivalent	`true`
DOS equivalent	None Applicable

18. Commands U-Z

18.1 *uname* - Display System Information

Synopsis

```
uname
```

Description

Displays data about the system. If no options are specified, the default is -s.

UNIX equivalent	`uname`
DOS equivalent	`ver`

Options

-a	Equivalent to *-snrvm*.
-c	Display a copyright message.
-m	Display the processor type name.
-n	Display the number of this processor within the intent network.
-r	Display the release level of the operating system.
-s	Display the name of the operating system.
-v	Display the version number of the operating system.

18.2 *unexpand* - Replaces leading whitespace with TAB characters.

Synopsis

```
unexpand
```

Description

Copies standard input to standard output, replacing leading whitespace on each line with TAB characters as much as possible.

Options

-a	In addition to leading whitespace, process every sequence of two or more whitespace characters anywhere on the line.
-t <tablist>	Specifies where tabstops are located. <tablist> is a whitespace- or comma-separated list of decimal column numbers. If only one number is given, Tabstops are separated by that many columns (default 8). If more than one number is given, Tabstops are at the named columns, and every column after the last specified. Tabstops must be listed in increasing order, and column zero is not valid. -t also implies -a.

18.3 *uniq* - Produce Listing of Identical Lines

Synopsis

```
uniq [<input-file> [<output-file>]]
```

Description

Lines are read from the specified input file (standard input by default). Sequences of consecutive identical lines are recognised and compressed, and the resulting sequence of unique lines is written to the specified output file (standard output by default).

Options

-c	At the beginning of each output line, output a decimal number indicating how many consecutive copies of the line were in the input.
-d	Output only duplicate lines. That is, do not output lines that appeared only once in the input.
-f <n>	When comparing lines, ignore the first <n> fields of each line (default zero). A field is defined to be a maximal sequence of blank characters followed by a maximal sequence of non-blank characters. (Space and tab characters are the blank characters.)
-s <n>	When comparing lines, ignore the first <n> characters (default zero), after skipping fields.
-u	Output only unique lines. That is, do not output lines that appeared more than once in the input.

18.4 *unset* - Delete Variables

Synopsis

```
unset <var> ...
```

Description

Deletes the specified shell variables.

UNIX equivalent	`unset`
DOS equivalent	None Applicable

18.5 *uuencode* - Encode File

Synopsis

```
uuencode [<input-filename>] <decode-filename>
```

Description

Encodes the specified input file (standard input if none is specified) in a standard textual form safe for transmission over text-only channels to other systems. The encoded file is written to standard output. The specified *<decode-filename>* will be included in the encoded output, as a suggestion of a suitable name for the encoded file.

The encoded file is approximately 35% larger than the original file. (A base-64 encoding is used - 3 octets are encoded as 4 characters - and there is some overhead in control information.)

UNIX® equivalent	uuencode
DOS® equivalent	None Applicable

18.6 *uudecode* - Decode Uuencode Output

Synopsis

```
uudecode [<input-filename>]
```

Description

Decodes the output of uuencode, saving the decoded file using the pathname stored in *uuencode*'s output

18.7 *vdu* - Bidirectional I/O between *stdio* and a device.

Synopsis

```
vdu [<options>]
```

Description

Sends characters from *stdin* to a device (by default */device/serial*) and sends characters received from the device to stdout. Certain characters are treated as commands to *vdu* itself and are not sent to the device. These are:

Ctrl-D	Exit vdu.
Alt-H	Hang-up connection. This causes vdu to cease communicating with the device and close the driver. After a brief pause it then reopens the device if possible.
Alt-B	Change the baud rate. The user is prompted to enter the new baud rate, which is then treated as if provided via the '-b' option.
Alt-S	Send the contents of a file to the device. The user is prompted for a filename.
Alt-X	Output a byte specified in hex to the device. The user is prompted for the hex code.
Alt-R	Redirect data from the device to the logfile so that it does not appear on-screen at all. This command toggles redirection on/off.
Alt-F	Set log file name. The user is prompted for a filename. This is then used to log data from the device exactly as if the -f option had been used. If the filename is empty then logging is switched off and data is subsequently sent to stdout.

Options

-a	Prevents the stripping of control characters, which is otherwise carried out on all except \t and \n.
-b<baud>	Sets the baud rate to <*baud*> (only makes sense for some devices).
-c	Causes incoming carriage-returns to be translated into line-feeds.
-e	Causes characters sent to the device to be echoed to stdout.
-f<file>	Data from the device is appended to the file named <*file*> instead of being sent to stdout. <*file*> is created if it does not exist already.
-i<name>	Uses <*name*> as the device to talk to instead of */device/serial*.
-n	Suppresses the translation of line-feeds to carriage-returns, which is the default behaviour.
-r	This option redirects data from the device directly to the log file. It is ignored if logging is not activated. (See also Alt-R above).
-s	This option only applies when a log file is being used. It causes data to be sent to stdout as well as being appended to the file.
-w	This option puts the program into raw mode so that no processing or filtering is performed on input and output data. It is primarily intended to allow an interactive shell to be controlled in the normal way over a character device.
-x	Displays characters received from the device as hex numbers.

18.8 *wc* - Print the Number of Bytes, Characters, Words, and Lines in Files

Synopsis

```
wc [<filename> ...]
```

Description

Count the number of bytes, characters, words and lines in the specified files, or standard input if no files are specified. Output the counts specified by options (by default, bytes, words and lines). If more than one filename is specified, total counts across all files are also displayed.

Options

-c	Output byte counts.
-m	Output (multibyte) character counts.
-w	Output word counts. A word is defined as a maximum-length sequence of consecutive non-

	whitespace characters.
-l	Output line counts. This is actually a count of newline characters.

UNIX® equivalent	wc
DOS® equivalent	None Applicable

18.9 *while* - Execute Command While Condition is True

Synopsis

```
while <condition> <command>
```

Description

Executes the specified condition as a command. If that exits with a zero exit status (indicating success), execute the specified command and then start again. This should be repeated until the condition fails. The status is the exit status of the last command executed.

It is possible to use the *break* and *continue* commands to modify the flow of control through a *while* loop.

UNIX equivalent	`while`
DOS equivalent	None Applicable

18.10 *xargs* - Build and Execute Command Lines From Standard Input

Synopsis

```
xargs [ <command> [<argument> ...] ]
```

Description

xargs provides a way to execute a command with more arguments than would comfortably fit into memory. The specified command is executed repeatedly, each time appending a limited number of arguments read from standard input, until the input is exhausted.

By default, the arguments added to the command line will be limited to a total length of 64kB. This criterion can be adjusted by options (see below).

If no <*command*> is specified, the default is *echo*. Note that only external commands - tools - can be executed directly. To execute a shell function or other special type of command, run, for example,

```
xargs -- shell -c {for -- a $shell.argv {foo -- $a}} --
```

instead of *xargs -- foo --*. (Note also that a "--" argument to xargs is necessary if you wish to include any options on the subordinate command line. A "--" as the last argument, terminating the subordinate command's options, is also necessary if any of the arguments on standard input may begin with a "-".)

By default, arguments on standard input are separated by whitespace, and shell-style quoting is understood. That is, an unquoted backslash character quotes the following character, and unquoted braces quote any enclosed text in which quoting is balanced.

UNIX equivalent	`xargs`
DOS equivalent	None Applicable

Exit status

0	Successful, and all commands invoked exited with status zero.
1	Runtime error.
2	Usage error.
3	Successful, but at least one command invoked returned a non-zero status.

Options

-0	Arguments on standard input are terminated by NUL bytes, rather than separated by whitespace. No quoting is possible. (*find*'s *-print0* option produces output suitable for this option.)
-n <number>	On each invocation of the command, use the specified number of arguments. (The last invocation may use fewer than this number, if there is insufficient input left.)
-P <number>	Run up to the specified number of copies of the specified command simultaneously. (The default is 1.) If 0 is specified, no such limit is imposed.
-r	Do not run the command at all if the input is empty. If this option is not specified, the command will be run once if the input is empty.
-s <size>	On each invocation of the command, use the maximum possible number of arguments such that their total length does not exceed the specified number of bytes. However, if any single argument exceeds this limit, it will be used anyway, as the only added argument for that invocation.

Efficiency

xargs is much faster when using the -0 option than when not using it. When using *find*, therefore, always use the *-print0* option for efficiency, even if it is not actually required due to special characters in pathnames.

Size-limited argument blocks (the -s option, and default behaviour) are processed slightly more efficiently than number-limited argument blocks (-n).

For alignment reasons, a buffer size (-s) that is a power of two (or a multiple of a large power of two) has some advantage over an unaligned size. Also, because the specified size is used for I/O blocking, a very small size will result in a large amount of I/O overhead.

18.11 *xor* - Combine Two Input Streams

Synopsis

```
xor
```

Description

xor combines two input streams via a bitwise exclusive-or, sending the resulting stream to standard output.

The two streams to be combined are standard input (file descriptor 0) and file descriptor 3. Standard input is regarded as being the primary input stream.

The output will have exactly the same length as the primary input. It is an error for the secondary input stream to be shorter than the primary input stream. The secondary input stream will not be read beyond the length required to match the primary input stream.

18.12 *yes* - Repeatedly Copy Arguments to Stdout

Synopsis

```
yes [<arg> ...]
```

Description

Repeatedly copies its arguments to standard output, separated by spaces and followed by a newline. If no arguments are provided, the default string is "y".

yes does not terminate normally. If successful, it will eventually receive a SIGPIPE (when the output stream is closed), upon which it exits with status zero.

Options

-n	Suppress the implicit newlines.

© Tao Group Ltd or Tao Systems Ltd. 2000, 2001. All Rights Reserved.

Copyright in the software either belongs to Tao Group Ltd or Tao Systems Ltd. The software may not be used, sold, licensed, transferred, copied or reproduced in whole or in part or in any manner or form other than in accordance with the licence agreement provided with the software or otherwise without the prior written consent of either Tao Group Ltd or Tao Systems Ltd.

No part of this publication may be reproduced in any material form (including photocopying or storing it in any medium by electronic means and whether or not transiently or incidentally to some other use of this publication) without the written permission of the copyright owner.

*Elate®, int**e**nt® and the Tao logo are registered trademarks of Tao Group Ltd.*
Digital Heaven™ is a trademark of Tao Group Ltd.
The rights of third party trademark owners are acknowledged.