1 `test`: extract from POSIX definition of the `test` command

NAME

test - evaluate expression

SYNOPSIS

```
test [expression]

[ [expression] ]
```

DESCRIPTION

The test utility shall evaluate the expression and indicate the result of the evaluation by its exit status. An exit status of zero indicates that the expression evaluated as true and an exit status of 1 indicates that the expression evaluated as false.

In the second form of the utility, which uses "`[]`" rather than test, the application shall ensure that the square brackets are separate arguments.

OPERANDS

The application shall ensure that all operators and elements of primaries are presented as separate arguments to the test utility.

The following primaries can be used to construct expression:

`-b file` True if `file` exists and is a block special file.

`-c file` True if `file` exists and is a character special file.

`-d file` True if `file` exists and is a directory.

`-e file` True if `file` exists.

`-f file` True if `file` exists and is a regular file.

`-g file` True if `file` exists and its set-group-ID flag is set.

`-h file` True if `file` exists and is a symbolic link.

`-L file` True if `file` exists and is a symbolic link.

`-n string` True if the length of `string` is non-zero.

`-p file` True if `file` is a FIFO.

`-r file` True if `file` exists and is readable. True shall indicate that permission to read from `file` will be granted, as defined in File Read, Write, and Creation.

`-S file` True if `file` exists and is a socket.

`-s file` True if `file` exists and has a size greater than zero.

`-t file_descriptor` True if the file whose file descriptor number is `file_descriptor` is open and is associated with a terminal.

`-u file` True if `file` exists and its set-user-ID flag is set.

`-w file` True if `file` exists and is writable. True shall indicate that permission to write from `file` will be granted, as defined in File Read, Write, and Creation.

`-x file` True if `file` exists and is executable. True shall indicate that permission to execute `file` will be granted, as defined in File Read, Write, and Creation. If `file` is a directory, true shall indicate that permission to search `file` will be granted.

`-z string` True if the length of string `string` is zero.

`string` True if the string `string` is not the null string.

`s1 = s2` True if the strings `s1` and `s2` are identical.

`s1 != s2` True if the strings `s1` and `s2` are not identical.

`n1 -eq n2` True if the integers `n1` and `n2` are algebraically equal.

`n1 -ne n2` True if the integers `n1` and `n2` are not algebraically equal.

`n1 -gt n2` True if the integer `n1` is algebraically greater than the integer `n2`.

`n1 -ge n2` True if the integer `n1` is algebraically greater than or equal to the integer `n2`.

`n1 -lt n2` True if the integer `n1` is algebraically less than the integer `n2`.

`n1 -le n2` True if the integer `n1` is algebraically less than or equal to the integer `n2`.

`expression1 -a expression2` True if both `expression1` and `expression2` are true. The `-a` binary primary is left associative. It has a higher precedence than `-o`.

`expression1 -o expression2` True if either `expression1` or `expression2` is true. The `-o` binary primary is left associative.

2 Numeric File Permissions: an Excerpt from the GNU Documentation

File permissions are stored internally as 16 bit integers. As an alternative to giving a symbolic mode, you can give an octal (base 8) number that corresponds to the internal representation of the new mode. This number is always interpreted in octal; you do not have to add a leading 0, as you do in C. Mode 0055 is the same as mode 55.

A numeric mode is usually shorter than the corresponding symbolic mode, but it is limited in that it can not take into account a file's previous permissions; it can only set them absolutely.

The permissions granted to the user, to other users in the file's group, and to other users not in the file's group are each stored as three bits, which are represented as one octal digit. The three special permissions are also each stored as one bit, and they are as a group represented as another octal digit. Here is how the bits are arranged in the 16 bit integer, starting with the lowest valued bit:

Value in Mode	Corresponding Permission
	Other users not in the file's group:
1	Execute
2	Write
4	Read
	Other users in the file's group:
10	Execute
20	Write
40	Read
	The file's owner:
100	Execute
200	Write
400	Read
	Special permissions:
1000	Save text image on swap device
2000	Set group ID on execution
4000	Set user ID on execution

For example, numeric mode 4755 corresponds to symbolic mode '`u=rwxs,go=rx`', and numeric mode 664 corresponds to symbolic mode '`ug=rw,o=r`'. Numeric mode 0 corresponds to symbolic mode '`ugo=`'.

3 `sed`: an Excerpt from Lecture Notes

`sed`—the Stream Editor

- `sed` provides many facilities for editing files
- `sed` reads from standard input and writes to standard output
- The *substitute* command, `s///`, is the most important
- The syntax (using `sed` as an editor of standard input), is:

  `$ sed 's/`*`original`*`/`*`replacement`*`/'`
- Example: replace the first instance of `Windows` with `Linux` on each line of the input:

  `sed 's/Windows/Linux/'`
- Example: replace *all* instances of `Windows` with `Linux` on each line of the input:

  `sed 's/Windows/Linux/g'`

4 Syntax of Some POSIX Commands

`find` [⟨*path*⟩ ...] [⟨*expression*⟩]

`find` searches the directory tree rooted at each ⟨*path*⟩ by evaluating ⟨*expression*⟩ from left to right. The first argument that begins with '`-`', '`(`', '`)`', '`,`', or '`!`' is taken to be the beginning of the ⟨*expression*⟩; any arguments before it are ⟨*path*⟩s to search, and any arguments after it are the rest of the ⟨*expression*⟩.

Expressions: The expression is made up of *options* (which affect overall operation rather than the processing of a specific file, and always return true), *tests* (which return a true or false value), and *actions* (which have side effects and return a true or false value), all separated by operators. `-and` is assumed where the operator is omitted. If the expression contains no actions, `-print` is performed on all files for which the expression is true.

Tests:

`-name` ⟨*pattern*⟩ Base of file name (the path with the leading directories removed) matches shell pattern ⟨*pattern*⟩.

Actions:

`-exec` ⟨*command*⟩ `;`
: Execute ⟨*command*⟩; true if 0 status is returned. All following arguments to find are taken to be arguments to the command until an argument consisting of '`;`' is encountered. The string '{}' is replaced by the current file name being processed. Both of these constructions might need to be escaped (with a '\') or quoted to protect them from expansion by the shell. The command is executed in the starting directory.

`-print` True; print the full file name on the standard output, followed by a newline.

`xargs` [⟨*options*⟩] [⟨*command*⟩ [⟨*initial-arguments*⟩]]

`xargs` reads arguments from the standard input, delimited by blanks (which can be protected with double or single quotes or a backslash) or newlines, and executes the command (default is `/bin/echo`) one or more times with any ⟨*initial-arguments*⟩ followed by arguments read from standard input. Blank lines on the standard input are ignored.

Options:

`--replace`, `-i` Replace occurences of {} in the initial arguments with names read from standard input. Also, unquoted blanks do not terminate arguments.

`egrep` [⟨*options*⟩] ⟨*pattern*⟩ [⟨*file*⟩...]
`grep` [⟨*options*⟩] ⟨*pattern*⟩ [⟨*file*⟩...]

`grep` searches the named input ⟨*file*⟩s (or standard input if no files are named, or the file name - is given) for lines containing a match to the given ⟨*pattern*⟩. By default, grep prints the matching lines.

Options:

`-E`, `--extended-regexp` Interpret ⟨*pattern*⟩ as an extended regular expression. `egrep` is the same as `grep -E`.

`-c`, `--count` Suppress normal output; instead print a count of matching lines for each input file. With the `-v`, `--invert-match` option (see below), count non-matching lines.

`-v`, `--invert-match` Invert the sense of matching, to select non-matching lines.

`sort` [⟨*option*⟩...] [⟨*file*⟩...]

Write sorted concatenation of all ⟨*file*⟩(s) to standard output.

Options:

`-b`, `--ignore-leading-blanks` ignore leading blanks

`-f`, `--ignore-case` fold lower case to upper case characters

`-k`⟨*pos1*⟩`[,`⟨*pos2*⟩`]`, `--key=`⟨*pos1*⟩`[,`⟨*pos2*⟩`]` start a key at ⟨*pos1*⟩, end it at ⟨*pos2*⟩ (column numbers start with 1)

`-r`, `--reverse` reverse the result of comparisons

`-n`, `--numeric-sort` compare according to string numerical value

`-t` ⟨*sep*⟩, `--field-separator=`⟨*sep*⟩

`diff` [⟨*options*⟩] ⟨*from-file*⟩ ⟨*to-file*⟩

In the simplest case, diff compares the contents of the two files ⟨*from-file*⟩ and ⟨*to-file*⟩.

`-u` Use the unified output format.

`cut` [⟨*option*⟩...] [⟨*file*⟩...]

Print selected parts of lines from each FILE to standard output.

`-b`, `--bytes=`⟨*list*⟩ output only these bytes

`-c`, `--characters=`⟨*list*⟩ output only these characters

`-d` ⟨*delim*⟩, `-delimiter=`⟨*delim*⟩ use ⟨*delim*⟩ instead of TAB for field delimiter

`-f` ⟨*list*⟩, `--fields=`⟨*list*⟩ output only these fields; also print any line that contains no delimiter character

Use one, and only one of `-b`, `-c` or `-f`. Each ⟨*list*⟩ is made up of one *range*, or many *ranges* separated by commas. Each *range* is one of:

N *N*th byte, character or field, counted from 1

***N*-** from *N*th byte, character or field, to end of line

N*-*M from *N*th to *M*th (included) byte, character or field

-*M* from first to *M*th (included) byte, character or field

`file` [⟨*option*⟩`...`] ⟨*file*⟩`...`

Identify the type of a file by examining the beginning of the file, and comparing with patterns in the "*magic*" file `/usr/share/file/magic`.

Options:

`-b` Do not prepend filenames to output lines (brief mode).

5 `awk`: an Excerpt from Lecture notes

Basic `awk`

- `awk` is a complete programming language
- Mostly used for one-line solutions to problems of extracting columns of data from text, and processing it

What Does `awk` Do?

- `awk` reads file(s) or standard input one line at a time, and
- automatically splits the line into fields, and calls them `$1`, `$2`,..., `$NF`
- `NF` is equal to the number of fields the line was split into
- `$0` contains the whole line
- `awk` has an option `-F` that allows you to select another pattern as the field separator
  - Normally `awk` splits columns by white space
- To execute code after all lines are processed, create an `END` block.

`awk` Examples

- Print the sizes of all files in current directory:

```
$ ls -l | awk '{print $5}'
```

- Add the sizes of all files in current directory and print the size:

```
$ ls -l | awk '{sum += $5} END{print sum}'
```

- Print only the permissions, user, group and file names of files in current directory:

```
$ ls -l | awk '{print $1, $3, $4, $NF}'
```

6 Extract from manual page for crontab file format

A *crontab* file contains instructions to the cron(8) daemon of the general form: "run this command at this time on this date". Each user has their own crontab, and commands in any given crontab will be executed as the user who owns the crontab.

Blank lines and leading spaces and tabs are ignored. Lines whose first non-space character is a pound-sign (`#`) are comments, and are ignored. Note that comments are not allowed on the same line as cron commands, since they will be taken to be part of the command.

cron(8) examines cron entries once every minute.

The time and date fields are:

field	**allowed values**
minute	0–59
hour	0–23
day of month	1–31
month	1–12 (or names, see below)
day of week	0–7 (0 or 7 is Sun, or use names)

A field may be an asterisk (`*`), which always stands for "first-last".

Ranges of numbers are allowed. Ranges are two numbers separated with a hyphen. The specified range is inclusive. For example, `8-11` for an "hours" entry specifies execution at hours 8, 9, 10 and 11.

Lists are allowed. A list is a set of numbers (or ranges) separated by commas. Examples: "`1,2,5,9`", "`0-4,8-12`".

Step values can be used in conjunction with ranges. Following a range with "/⟨*number*⟩" specifies skips of the ⟨*number*⟩'s value through the range. For example, "`0-23/2`" can be used in the hours field to specify command execution every other hour (the alternative in the V7 standard is "`0,2,4,6,8,10,12,14,16,18,20,22`"). Steps are also permitted after an asterisk, so if you want to say "every two hours", just use "`*/2`".

Names can also be used for the "month" and "day of week" fields. Use the first three letters of the particular day or month (case doesn't matter). Ranges or lists of names are not allowed.

The "sixth" field (the rest of the line) specifies the command to be run. The entire command portion of the line, up to a newline or % character, will be executed by `/bin/sh` or by the shell specified in the `SHELL` variable of the cronfile. Percent-signs (`%`) in the command, unless escaped with backslash (\), will be changed into newline characters, and all data after the first `%` will be sent to the command as standard input.

Note: The day of a command's execution can be specified by two fields day of month, and day of week. If both fields are restricted (ie, aren't `*`), the command will be run when either field matches the current time. For example, "`30 4 1,15 * 5`" would cause a command to be run at 4.30 am on the 1st and 15th of each month, plus every Friday.

EXAMPLE CRON FILE

```
# run five minutes after midnight, every day
5 0 * * *       $HOME/bin/daily.job >> $HOME/tmp/out 2>&1
# run at 2:15pm on the first of every month
15 14 1 * *     $HOME/bin/monthly
```

```
# run at 10 pm on weekdays, annoy Joe
0 22 * * 1-5   mail -s "It's 10pm" joe%Joe,%%Where are your kids?%
23 0-23/2 * * * echo "run 23 minutes after midn, 2am, 4am ..., everyday"
5 4 * * sun     echo "run at 5 after 4 every sunday"
```

7 Examples of using rsync over SSH

In these examples, the options `-avz` mean:

`-a`, `--archive` is a simple way of saying you want to copy recursively (i.e., all subdirectories), and preserve all permissions, ownerships and time stamps.

`-v`, `--verbose` increase the amount of information given; one `-v` will print the name of each file as it is transferred, and a brief summary at the end.

`-z`, `--compress` Compress the data sent to the destination.

`-e ssh` tells `rsync` to transfer the data over secure shell (SSH).

To transfer content of directory, but not the directory itself, from `/usr/src/` on the local machine to the directory `/usr/local/sources` on host `ictlab`

```
$ rsync -avz -e ssh /usr/src/ ictlab:/usr/local/sources
```

So for example, a file `/usr/src/file.c` on the local machine will be transferred to the name `/usr/local/sources/file.c` on the host `ictlab`. Note the trailing slash on the source directory name.

To transfer the directory `/usr/src` itself, from the local machine to the directory `/usr/local/sources` on host `ictlab`

```
$ rsync -avz -e ssh /usr/src ictlab:/usr/local/sources
```

So for example, a file `/usr/src/file.c` on the local machine will be transferred to the name `/usr/local/src/sources/file.c` on the host `ictlab`. Note there is no trailing slash on the source directory name.

8 Help Output for some `bash` Builtin Commands

`trap` [⟨*arg*⟩] [⟨*signal spec*⟩ ...]

The command ⟨*arg*⟩ is to be read and executed when the shell receives signal(s) ⟨*signal spec*⟩. If ⟨*arg*⟩ is absent all specified signals are reset to their original values. If ⟨*arg*⟩ is the null string each ⟨*signal spec*⟩ is ignored by the shell and by the commands it invokes. If a ⟨*signal spec*⟩ is `EXIT` (0) the command ⟨*arg*⟩ is executed on exit from the shell. If a ⟨*signal spec*⟩ is `DEBUG`, ⟨*arg*⟩ is executed after every command. If ⟨*arg*⟩ is '`-p`' then the trap commands associated with each ⟨*signal spec*⟩ are displayed. If no arguments are supplied or if only '`-p`' is given, trap prints the list of commands associated with each signal number. Each ⟨*signal spec*⟩ is either a signal name in `<signal.h>` or a signal number. Note that a signal can be sent to the shell with "`kill -signal $$`".

`jobs [-lnprs]` [⟨*jobspec*⟩ ...]

Lists the active jobs. The `-l` option lists process IDs in addition to the normal information; the `-p` option lists process IDs only. If `-n` is given, only processes that have changed status since the last notification are printed. ⟨*jobspec*⟩ restricts output to that job. The `-r` and `-s` options restrict output to running and stopped jobs only, respectively. Without options, the status of all active jobs is printed.

`fg` [⟨*jobspec*⟩]

Place ⟨*jobspec*⟩ in the foreground, and make it the current job. If ⟨*jobspec*⟩ is not present, the shell's notion of the current job is used.

`bg` [⟨*jobspec*⟩]

Place ⟨*jobspec*⟩ in the background, as if it had been started with '`&`'. If ⟨*jobspec*⟩ is not present, the shell's notion of the current job is used.

9 Regular Expression Metacharacters

`\`	Quote the next metacharacter
`^`	Match the beginning of the line
`.`	Match any character (except newline)
`$`	Match the end of the line (or before newline at the end)
`|`	Alternation
`(...)`	Grouping, and captures so that it can be used as a backreference
`[...]`	Character class
`[^...]`	Negated character class
	Match 0 or more times
`+`	Match 1 or more times
`?`	Match 1 or 0 times
{n}	Match exactly n times
{n,}	Match at least n times
{n,m}	Match at least n but not more than m times