NAME

bfs − big file scanner

SYNOPSIS

bfs [−] name

DESCRIPTION

Bfs is (almost) like *ed*(I) except that it is read-only and processes much bigger files. Files can be up 1024K bytes (the maximum possible size) and 32K lines, with up to 255 characters per line. *Bfs* is usually more efficient than *ed* for scanning a file, since the file is not copied to a buffer.

Normally, the size of the file being scanned is printed, as is the size of any file written with the *w* command. The optional − suppresses printing of sizes. Input is prompted with '*' if 'P' and a carriage return is typed as in *ed*. Prompting can be turned off again by inputting another 'P' and carriage return. Note that messages are given in response to errors if prompting is turned on.

All address expressions described under *ed* are supported. In addition, regular expressions may be surrounded with two symbols besides '/' and '?': '>' indicates downward search without wrap-around, and '<' indicates upward search without wrap-around. Since *bfs* uses a different regular expression-matching routine from *ed*, the regular expressions accepted are slightly wider in scope (see *regex*(III)). There is a slight difference in mark names: only the letters 'a' through 'z' may be used, and all 26 marks are remembered.

The *e, g, v, k, n, p, q, w, =, !* and null commands operate as described under *ed*. Commands such as '−−−', '+++−', '+++=', '−12', and '+4p' are accepted. Note that '1,10p' and '1,10' will both print the first ten lines. The *f* command only prints the name of the file being scanned; there is no *remembered* file name. The *w* command is independent of output diversion, truncation or crunching (see the *xo, xt* and *xc* commands, below). The following additional commands are available:

xf file

Further commands are taken from the named file. When an end-of-file is reached, an interrupt signal is received or an error occurs, reading resumes with the file containing the *xf*. *Xf* commands may be nested to a depth of 10.

xo [file]

Further output from the *p* and null commands is diverted to the named file, which, if necessary, is created mode 666. Plain 'xo' diverts output back to the standard output. Note that each diversion causes truncation or creation of the file.

: label

This positions a label in a command file. The label is terminated by newline, and blanks between the ':' and the start of the label are ignored. This command may also be used to insert comments into a command file, since labels need not be referenced.

(**.** , **.**) xb/regular expression/label

A jump (either upward or downward) is made to the named label if the command succeeds. It fails under any of the following conditions:

1. Either address is not between 1 and $.
2. The second address is less than the first.
3. The regular expression doesn't match at least one line in the specified range, including the first and last lines.

On success, '.' is set to the line matched and a jump is made to the label. This command is the only one that doesn't issue an error message on bad addresses, so it may be used to test whether addresses are bad before other commands are executed. Note that the command

```
xb/^/ label
```

is an unconditional jump.

The *xb* command is allowed only if it is read from someplace other than a terminal. If it is read from a pipe only a downward jump is possible.

xt number

Output from the *p* and null commands is truncated to at most *number* characters. The initial number is 255.

xv[digit: 0−9][optional spaces][value]

The variable name is the specified *digit* following the ‘xv’. ‘xv5100’ or ‘xv5 100’ both assign the *value* ‘100’ to the *variable* ‘5’. ‘xv61,100p’ assigns the *value* ‘1,100p’ to *variable* ‘6’. To reference the variable put a ‘%’ in front of the variable name. For example, using the above assignments for the variables ‘5’ and‘6’:

```
1,%5p
1,%5
%6
```

will all print the first 100 lines.

```
g/%5/p
```

would globally search for the characters ‘100’ and print each line containing a match. To escape the special meaning of ‘%’, a ‘\’ must precede it.

```
g/".*\%[cds]/p
```

could be used to match and list lines containing *printf* of characters,decimal integers, or strings.

Another feature of the *xv* command is that the first line of output from a UNIX command can be stored into a variable. The only requirement is that the first character of *value* be an ‘!’. For example:

```
.w junk
xv5!cat junk
!rm junk
!echo "%5"
xv6!expr %6 + 1
```

would put the current line into variable ‘5’, print it, and increment the variable ‘6’ by one. To escape the special meaning of ‘!’ as the first character of *value,* precede it with a ‘\’.

```
xv7\!date
```

stores the value ‘!date’ into variable ‘7’.

xbz label
xbn label

These two commands will test the last saved *return code* from the execution of a unix command (!UNIX command) and branch on a zero or nonzero value, respectively, to the specified label. The two examples below both search for the next five lines containing the string ‘size’.

```
xv55
: l
/size/
xv5!expr %5 - 1
!if 0%5 != 0 exit 2
xbn l
```

```
xv45
: l
/size/
xv4!expr %4 - 1
!if 0%4 = 0 exit 2
xbz l
```

xc [switch]

If *switch* is 1, output from the *p* and null commands is crunched; if *switch* is 0 it isn't. Plain 'xc' reverses the switch. Initially the switch is set for no crunching. Crunched output has strings of tabs and blanks reduced to one blank and blank lines suppressed.

SEE ALSO

ed(I), regex(III)

DIAGNOSTICS

'?' for errors in commands, if prompting is turned off. Self-explanatory error messages when prompting is on.