NAME

sed - stream editor

SYNOPSIS

sed [**-g**] [**-n**] [**-f** commandfile] ... [[**-e**] command] ... [file] ...

DESCRIPTION

Sed copies the input files (standard input default) to the standard output, perhaps performing one or more editor commands (see ed (I)) on each line.

The following flags are interpreted by sed :

-e Indicates that the next argument is an editor command.

-f Indicates that the next argument is a filename; the file contains editor commands, one to a line. Commands which are inherently multi-line, like a or c, should be written with the interior newlines preceded by `\'. Append mode is terminated by an unhidden newline.

The **-e** and **-f** flags may be intermixed in any order. If no **-e** or **-f** flags are given, the first argument is taken by default to be an editor command.

Addresses are allowed. The meaning of two addresses is: "Attempt this command on the first line matching the first address, and on all subsequent lines until the next line containing a match of the second address; then begin watching for a match of the first address and iterate." One address means: "Attempt this command on all the lines which match the address." allowable as addresses. A line number matches the cumulative line number of the input files. A `$' as an address matches the last line of the last input file.

-g Indicates that all s commands should be executed as though followed by a g. If only some substitutes are to be done globally, leave out the **-g** flag, and put the g's at the end of the appropriate command lines.

-n Prevents the copying of lines to the output by default. Only lines which are explicitly printed by p commands are written. In order to avoid getting double copies of some lines, the p command is a no-op unless the **-n** flag is set.

The intention is to simulate the editor as exactly as possible, but the line-at-a-time operation makes certain differences una-

voidable or desirable:

1. There is no notation '.'; and no relative addressing.

2. Commands with no addresses are defaulted to 1,$ rather than to dot.

3. Context addresses must be delimited by '/'; '?' is an error.

4. Expressions in addresses are not allowed.

5. Commands may have only as many addresses as they can use. That is, no command may have more than two addresses; the a, i, r, and l commands may have only one address.

6. A p at the end of a command only works with the s command. For other commands, or if the -n flag is not in effect, a p at the end of a command line is a no-op.

7. A w may appear at the end of a substitute command. It should be followed by a single space and a file name. If the substitute command is successfully executed, the line is written to the file. All files are opened when the commands are being compiled, and closed when the program terminates. Only ten different file names may appear in w commands in a single run. Unlike p, w takes effect regardless of the -n flag. If both p and w are appended to the same substitute command, they must be in the order pw.

8. The only commands available are a, c, d, i, s, p, q, r, w, g, v, and =. A successful execution of a q command causes the current line to be written out if it should be, and execution terminated. When a line is deleted by a d or c command, no further commands are attempted on the corpse, but another line is immediately read from the input (but see the next point (9)).

9. If an a, i, or r command is successfully executed, the text is inserted into the file whether or not the line on which the match was made is later deleted or not. Thus the commands:

```
/b/a\
XXX
/b/,/c/d
```

applied to the file

```
a
b
c
d
```

will produce

```
a
XXX
d
```

on the output.

10. Text inserted in the output stream by the a, i, c or r commands is not scanned for any pattern matches, nor are any editor commands applied to it.

SYNTAX:

Blank lines, blanks, and tabs at the beginning of a line in the command file are completely ignored.

Commands may be grouped by curly braces. The opening brace must appear in the place where a function would ordinarily appear; the closing brace must appear on a line by itself (except, of course, for leading blanks or tabs).

If the first line of a command file has #n as its first two characters, the no-copy flag is set, as though the **-n** option had been given on the command line. The rest of the line is ignored; it may be used for a title or a comment.

PATTERN MATCH COMMANDS:

These three capital letter commands are intended to allow pattern matches across new-line characters in the input file.

N Next line is appended to the current line; the two lines are separated by a new-line character which may be matched by `\n' (see below).

D Delete up to and including first (leftmost) new-line in the current pattern space. If the pattern space becomes empty (the only new-line was at the end of the space), read another line from the input. In any case, begin the list of editing commands again from the beginning.

P Print up to and including the first new-line in the pattern space.

META-CHARACTERS:

\n Matches imbedded newlines in the pattern space.

\\ Matches '\'

\] Matches ']'

Examples:

The file /mnt/btl/dir contains telephone-book entries. Most are on a single line; double line entries are signaled by having the second line begin with a blank.

To print out all double-line entries:

```
sed -n
    N
    /\n/p
    D /mnt/btl/dir
```

To combine all double line entries into single lines:

```
sed -n
    N
    /\n/{
        s// /p
        j (see flow-of-control commands)
    }
    P
    D /mnt/btl/dir >newdir
```

NEXT LINE COMMAND:

n The current line is replaced by the next line from the input file, and the list of editing functions is continued after the n function.

FLOW-OF-CONTROL COMMANDS:

These commands do no editing on the input line, but serve to control the order in which multipule editing commands are applied to an input line.

:<label>

The label command marks a place in the list of editing commands which may be referred to by j and t commands. The <label> may be any sequence of eight or fewer characters; if two different colon commands have identical labels, a compile time diagnostic will be generated, and no execution attempted.

j <label>

The jump command causes the sequence of editing commands being applied to the current input line to be restarted immediately after the place where a colon command with the same <label> was encountered. If no colon command with the same label can be found after all the editing commands have been compiled, a compile time diagnostic is produced, and no execution is at tempted.

A j command with no <label> is taken to be a jump to the end of the list of editing commands; whatever should be done with the current input line is done, and another input line is read; the list of editing commands is restarted from the beginning on the new line.

t <label>

The t command tests whether any successsful substitutions have been made on the current input line; if so, it jumps to <label>; if not, it does nothing. The flag which indicates that a successful substitution has been executed is reset by:

1) reading a new input line, and

2) executing a t command.

Example:

```
#n   This file prints each non-blank line following  a  blank
line.
1{
     /./p
}
/^$/{
: loop
     n
     /./{
          p
          j
     }
     j loop
}
```

FILES

SEE ALSO

ed (1)

DIAGNOSTICS

BUGS

Lines are silently truncated at 512 characters.