NAME

psed - a stream editor

SYNOPSIS

```
psed [-an] script [file ...]
psed [-an] [-e script] [-f script-file] [file ...]

s2p  [-an] [-e script] [-f script-file]
```

DESCRIPTION

A stream editor reads the input stream consisting of the specified files (or standard input, if none are given), processes is line by line by applying a script consisting of edit commands, and writes resulting lines to standard output. The filename `-' may be used to read standard input.

The edit script is composed from arguments of **-e** options and script-files, in the given order. A single script argument may be specified as the first parameter.

If this program is invoked with the name *s2p*, it will act as a sed-to-Perl translator. See *sed Script Translation*.

sed returns an exit code of 0 on success or >0 if an error occurred.

OPTIONS

-a

A file specified as argument to the **w** edit command is by default opened before input processing starts. Using **-a**, opening of such files is delayed until the first line is actually written to the file.

-e *script*

The editing commands defined by *script* are appended to the script. Multiple commands must be separated by newlines.

-f *script-file*

Editing commands from the specified *script-file* are read and appended to the script.

-n

By default, a line is written to standard output after the editing script has been applied to it. The **-n** option suppresses automatic printing.

COMMANDS

sed command syntax is defined as

[*address*[**,***address*]][**!**]*function*[*argument*]

with whitespace being permitted before or after addresses, and between the function character and the argument. The *address*es and the address inverter (`!`) are used to restrict the application of a command to the selected line(s) of input.

Each command must be on a line of its own, except where noted in the synopses below.

The edit cycle performed on each input line consist of reading the line (without its trailing newline character) into the *pattern space*, applying the applicable commands of the edit script, writing the final contents of the pattern space and a newline to the standard output. A *hold space* is provided for saving the contents of the pattern space for later use.

Addresses

A sed address is either a line number or a pattern, which may be combined arbitrarily to construct ranges. Lines are numbered across all input files.

Any address may be followed by an exclamation mark (`!'), selecting all lines not matching that address.

number

The line with the given number is selected.

$

A dollar sign ($) is the line number of the last line of the input stream.

/**regular expression**/

A pattern address is a basic regular expression (see *Basic Regular Expressions*), between the delimiting character `/`. Any other character except \ or newline may be used to delimit a pattern address when the initial delimiter is prefixed with a backslash (`\').

If no address is given, the command selects every line.

If one address is given, it selects the line (or lines) matching the address.

Two addresses select a range that begins whenever the first address matches, and ends (including that line) when the second address matches. If the first (second) address is a matching pattern, the second address is not applied to the very same line to determine the end of the range. Likewise, if the second address is a matching pattern, the first address is not applied to the very same line to determine the begin of another range. If both addresses are line numbers, and the second line number is less than the first line number, then only the first line is selected.

Functions

The maximum permitted number of addresses is indicated with each function synopsis below.

The argument *text* consists of one or more lines following the command. Embedded newlines in *text* must be preceded with a backslash. Other backslashes in *text* are deleted and the following character is taken literally.

[1addr]**a** *text*

Write *text* (which must start on the line following the command) to standard output immediately before reading the next line of input, either by executing the **N** function or by beginning a new cycle.

[2addr]**b** [*label*]

Branch to the **:** function with the specified *label*. If no label is given, branch to the end of the script.

[2addr]**c** *text*

The line, or range of lines, selected by the address is deleted. The *text* (which must start on the line following the command) is written to standard output. With an address range, this occurs at the end of the range.

[2addr]**d**

Deletes the pattern space and starts the next cycle.

[2addr]**D**

Deletes the pattern space through the first embedded newline or to the end. If the pattern space becomes empty, a new cycle is started, otherwise execution of the script is restarted.

[2addr]**g**

Replace the contents of the pattern space with the hold space.

[2addr]**G**

Append a newline and the contents of the hold space to the pattern space.

[2addr]**h**

Replace the contents of the hold space with the pattern space.

[2addr]**H**

Append a newline and the contents of the pattern space to the hold space.

[1addr]**i** *text*

Write the *text* (which must start on the line following the command) to standard output.

[2addr]**l**

Print the contents of the pattern space: non-printable characters are shown in C-style escaped form; long lines are split and have a trailing `\' at the point of the split; the true end of a line is marked with a `$'. Escapes are: `\a', `\t', `\n', `\f', `\r', `\e' for BEL, HT, LF, FF, CR, ESC, respectively, and `\' followed by a three-digit octal number for all other non-printable characters.

[2addr]**n**

If automatic printing is enabled, write the pattern space to the standard output. Replace the pattern space with the next line of input. If there is no more input, processing is terminated.

[2addr]**N**

Append a newline and the next line of input to the pattern space. If there is no more input, processing is terminated.

[2addr]**p**

Print the pattern space to the standard output. (Use the **-n** option to suppress automatic printing at the end of a cycle if you want to avoid double printing of lines.)

[2addr]**P**

Prints the pattern space through the first embedded newline or to the end.

[1addr]**q**

Branch to the end of the script and quit without starting a new cycle.

[1addr]**r** *file*

Copy the contents of the *file* to standard output immediately before the next attempt to read a line of input. Any error encountered while reading *file* is silently ignored.

[2addr]**s***Iregular expression***I***replacement***I***flags*

Substitute the *replacement* string for the first substring in the pattern space that matches the *regular expression*. Any character other than backslash or newline can be used instead of a slash to delimit the regular expression and the replacement. To use the delimiter as a literal character within the regular expression and the replacement, precede the character by a backslash (`\').

Literal newlines may be embedded in the replacement string by preceding a newline with a backslash.

Within the replacement, an ampersand (`&') is replaced by the string matching the regular expression. The strings `\1' through `\9' are replaced by the corresponding subpattern (see *Basic Regular Expressions*). To get a literal `&' or `\' in the replacement text, precede it by a backslash.

The following *flags* modify the behaviour of the **s** command:

g

The replacement is performed for all matching, non-overlapping substrings of the pattern space.

1..**9**

Replace only the n-th matching substring of the pattern space.

p

If the substitution was made, print the new value of the pattern space.

w *file*

If the substitution was made, write the new value of the pattern space to the specified file.

[2addr]**t** [*label*]

Branch to the **:** function with the specified *label* if any **s** substitutions have been made since the most recent reading of an input line or execution of a **t** function. If no label is given, branch to the end of the script.

[2addr]**w** *file*

The contents of the pattern space are written to the *file*.

[2addr]**x**

Swap the contents of the pattern space and the hold space.

[1addr]**=**

Prints the current line number on the standard output.

[0addr]**:** [*label*]

The command specifies the position of the *label*. It has no other effect.

[2addr]**{** [*command*]

[0addr]**}**

These two commands begin and end a command list. The first command may be given on the same line as the opening **{** command. The commands within the list are jointly selected by the address(es) given on the **{** command (but may still have individual addresses).

[0addr]**#** [*comment*]

The entire line is ignored (treated as a comment). If, however, the first two characters in the script are `#n', automatic printing of output is suppressed, as if the **-n** option were given on the command line.

BASIC REGULAR EXPRESSIONS

A *Basic Regular Expression* (BRE), as defined in POSIX 1003.2, consists of *atoms*, for matching parts of a string, and *bounds*, specifying repetitions of a preceding atom.

Atoms

The possible atoms of a BRE are: **.**, matching any single character; **^** and **$**, matching the null string at the beginning or end of a string, respectively; a *bracket expressions*, enclosed in **[** and **]** (see below); and any single character with no other significance (matching that character). A **** before one of: **.**, **^**, **$**, **[**, *****, ****, matching the character after the backslash. A sequence of atoms enclosed in **\(** and **\)** becomes an atom and establishes the target for a *backreference*, consisting of the substring that actually matches the enclosed atoms. Finally, **** followed by one of the digits **0** through **9** is a backreference.

A **^** that is not first, or a **$** that is not last does not have a special significance and need not be preceded by a backslash to become literal. The same is true for a **]**, that does not terminate a bracket expression.

An unescaped backslash cannot be last in a BRE.

Bounds

The BRE bounds are: *****, specifying 0 or more matches of the preceding atom; **\{***count***\}**, specifying that many repetitions; **\{***minimum***,\}**, giving a lower limit; and **\{***minimum***,***maximum***\}** finally defines a lower and upper bound.

A bound appearing as the first item in a BRE is taken literally.

Bracket Expressions

A *bracket expression* is a list of characters, character ranges and character classes enclosed in **[** and **]** and matches any single character from the represented set of characters.

A character range is written as two characters separated by **-** and represents all characters (according to the character collating sequence) that are not less than the first and not greater than the second. (Ranges are very collating-sequence-dependent, and portable programs should avoid relying on them.)

A character class is one of the class names

```
alnum     digit     punct
alpha     graph     space
blank     lower     upper
cntrl     print     xdigit
```

enclosed in **[:** and **:]** and represents the set of characters as defined in ctype(3).

If the first character after **[** is **^**, the sense of matching is inverted.

To include a literal `^', place it anywhere else but first. To include a literal ']' place it first or immediately after an initial **^**. To include a literal `-' make it the first (or second after **^**) or last character, or the second endpoint of a range.

The special bracket expression constructs `[[:<:]]` and `[[:>:]]` match the null string at the beginning and end of a word respectively. (Note that neither is identical to Perl's `\b' atom.)

Additional Atoms

Since some sed implementations provide additional regular expression atoms (not defined in POSIX 1003.2), **psed** is capable of translating the following backslash escapes:

\< This is the same as `[[:>:]]`.

\> This is the same as `[[:<:]]`.

\w This is an abbreviation for `[[:alnum:]_]`.

\W This is an abbreviation for `[^[:alnum:]_]`.

\y Match the empty string at a word boundary.

\B Match the empty string between any two either word or non-word characters.

To enable this feature, the environment variable PSEDEXTBRE must be set to a string containing the requested characters, e.g.: `PSEDEXTBRE='<>wW'`.

ENVIRONMENT

The environment variable `PSEDEXTBRE` may be set to extend BREs. See *Additional Atoms*.

DIAGNOSTICS

ambiguous translation for character `%s' in `y' command

The indicated character appears twice, with different translations.

`[' cannot be last in pattern

A `[' in a BRE indicates the beginning of a *bracket expression*.

`\' cannot be last in pattern

A `\' in a BRE is used to make the subsequent character literal.

`\' cannot be last in substitution

A `\' in a subsitution string is used to make the subsequent character literal.

conflicting flags `%s'

In an **s** command, either the `g' flag and an n-th occurrence flag, or multiple n-th occurrence flags are specified. Note that only the digits `1' through `9' are permitted.

duplicate label %s (first defined at %s)

excess address(es)

The command has more than the permitted number of addresses.

extra characters after command (%s)

illegal option `%s'

improper delimiter in s command

The BRE and substitution may not be delimited with `\' or newline.

invalid address after `,'

invalid backreference (%s)

The specified backreference number exceeds the number of backreferences in the BRE.

invalid repeat clause `\{%s\}'

The repeat clause does not contain a valid integer value, or pair of values.

malformed regex, 1st address

malformed regex, 2nd address

malformed regular expression

malformed substitution expression

malformed `y' command argument

The first or second string of a **y** command is syntactically incorrect.

maximum less than minimum in `\{%s\}'

no script command given

There must be at least one **-e** or one **-f** option specifying a script or script file.

`\' not valid as delimiter in `y' command

option -e requires an argument

option -f requires an argument

`s' command requires argument

start of unterminated `{'

string lengths in `y' command differ

The translation table strings in a **y** command must have equal lengths.

undefined label `%s'

unexpected `}'

A **}** command without a preceding **{** command was encountered.

unexpected end of script

The end of the script was reached although a text line after a **a**, **c** or **i** command indicated another line.

unknown command `%s'

unterminated `['

A BRE contains an unterminated bracket expression.

unterminated `\('

A BRE contains an unterminated backreference.

`\{' without closing `\}'

A BRE contains an unterminated bounds specification.

`\)' without preceding `\('

`y' command requires argument

EXAMPLE

The basic material for the preceding section was generated by running the sed script

```
#no autoprint
s/^.*Warn( *"\([^"]*\)".*$/\1/
t process
b
:process
s/$!/%s/g
s/$[_[:alnum:]]\{1,\}/%s/g
s/\\\\/\\/g
s/^/=item /
p
```

on the program's own text, and piping the output into `sort -u`.

SED SCRIPT TRANSLATION

If this program is invoked with the name *s2p* it will act as a sed-to-Perl translator. After option processing (all other arguments are ignored), a Perl program is printed on standard output, which will process the input stream (as read from all arguments) in the way defined by the sed script and the option setting used for the translation.

SEE ALSO

perl(1), re_format(7)

BUGS

The **l** command will show escape characters (ESC) as `\e', but a vertical tab (VT) in octal.

Trailing spaces are truncated from labels in **:**, **t** and **b** commands.

The meaning of an empty regular expression (`//'), as defined by **sed**, is "the last pattern used, at run time". This deviates from the Perl interpretation, which will re-use the "last last successfully executed regular expression". Since keeping track of pattern usage would create terribly cluttered code, and differences would only appear in obscure context (where other **sed** implementations appear to deviate, too), the Perl semantics was adopted. Note that common usage of this feature, such as in

`/abc/s//xyz/`, will work as expected.

Collating elements (of bracket expressions in BREs) are not implemented.

STANDARDS

This **sed** implementation conforms to the IEEE Std1003.2-1992 ("POSIX.2") definition of **sed**, and is compatible with the *OpenBSD* implementation, except where otherwise noted (see *BUGS*).

AUTHOR

This Perl implementation of *sed* was written by Wolfgang Laun, *Wolfgang.Laun@alcatel.at*.

COPYRIGHT and LICENSE