NAME

sam — screen editor with structural regular expressions

SYNOPSIS

sam [**-d**] [**-t** *terminal*] *file ...*
sam **-r** *machine* [**-s** *file*] [**-t** *terminal*] *file ...*
sam.save
B [**-r** *machine*] *file ...*

DESCRIPTION

sam is a multi-file editor. It modifies a local copy of an external file. The copy is here called a *file*. The files are listed in a menu available through mouse button 3 or the `n` command. Each file has an associated name, usually the name of the external file from which it was read, and a “modified” bit that indicates whether the editor’s file agrees with the external file. The external file is not read into the editor’s file until it first becomes the current file – that file to which editing commands apply – whereupon its menu entry is printed. The options are

-d Do not download the terminal part of **sam**. Editing will be done with the command language only, as in `ed`(1).

-r *machine*
Run the host part remotely on the specified machine, the terminal part locally, or cause commands to be sent to an instance of **sam** associated with *machine*.

-s *file*
Start the host part from the indicated file on the remote host. By default, this is `rsam`, which is a program that interposes itself between the host and terminal parts of **sam** and allows **sam** to be controlled via commands on the remote system.

-t *file*
Start the terminal part from the indicated file. Useful for debugging.

Regular expressions

Regular expressions are more-or-less as they are in `regex`(7), with the addition of `\n` to represent newlines. A regular expression may never contain a literal newline character. The elements of regular expressions are:

. Match any character except newline.

`\n` Match newline.

`\x` For any character except `n` match the character (here **x**).

`[abc]` Match any character in the square brackets. `\n` may be mentioned.

`[^abc]`
Match any character not in the square brackets, but never a newline. Both this and the positive form above accept a range of ASCII characters indicated by a dash, as in `a-z`.

^ Match the null string immediately after a newline.

`$` Match the null string immediately before a newline.

Any other character except newline matches itself.

In the following, **r1** and **r2** are regular expressions.

(**r1**) Match what **r1** matches.

r1|r2 Match what **r1** or **r2** matches.

r1* Match zero or more adjacent matches of **r1**.

r1+ Match one or more adjacent matches of **r1**.

r1? Match zero or one matches of **r1**.

The operators *, +, and ? are highest precedence, then catenation, then | is lowest. The empty regular expression stands for the last complete expression encountered. A regular expression in **sam** matches the longest leftmost substring formally matched by the expression. Searching in the reverse direction is equivalent to search backwards with the catenation operations reversed in the expression.

Addresses

An address identifies a substring in a file. In the following "character **n**" means the null string after the**n**-th character in the file, with 1 the first character in the file. "Line **n**" means the**n**-th match, starting at the beginning of the file, of the regular expression `.*\n?`. (The peculiar properties of a last line without a newline are temporarily undefined.) All files always have a current substring, called **dot**, that is the default address.

Simple addresses

# **n** The empty string after character **n**; `#0` is the beginning of the file.

n Line **n**.

/ **regexp** /

? **regexp** ?

The substring that matches the regular expression, found by looking toward the end (/) or be ginning ( )? of the file, and if necessary continuing the search from the other end to the starting point of the search. The matched substring may straddle the starting point. When entering a pattern containing a literal question mark for a backward search, the question mark should be specified as a member of a class.

0 The string before the first full line. This is not necessarily the null string; see + and - below.

$ The null string at the end of the file.

. Dot.

' The mark in the file (see the **k** command below) .

regexp "A regular expression in double quotes." Preceding a simple address "default .", refers to the address evaluated in the unique file whose menu line matches the regular expression.

Compound addresses

In the following, **a1** and **a2** are addresses.

a1+a2 The address **a2** evaulated starting at the end of **a1**.

a1-a2 The address **a2** evaluated looking the reverse direction starting at the beginning of **a1**.

a1,a2 The substring from the beinning of **a1** to the end of **a2**. If**a1** is missing, 0 is substituted. If **a2** is missing, $ is substituted.

a1;a2 Like "**a1,a2**" but with **a2** evaluated at the end of, and dot set to, **a1**.

The operators + and - are high precedence, while , and ; are low precedence.

In both `+` and `-` forms, if **a2** is a line or character address with a missing number, the number defaults to 1. If **a1** is missing, . is subtituted. If both **a1** and **a2** are present and distinguishable, `+` may be elided. **a2** may be a regular expression; if it is delimited by ? characters, the effect of the `+` or `-` is reversed.

It is an error for a compound address to represent a malformed substring.

Some useful idioms:

a1+- (**a1-+**)
selects the line containing the end “beginning” of**a1**.

0/regexp/
locates the first match of the expression in the file. “The form `0;//` sets dot unnecessarily.”

./regexp///
find the second following occurence of the expression, and **.,/regexp/** extends dot.

Commands

In the following, text demarcated by slashes represnets text delimited by any printable ASCII character except alphanumerics. Any number of trailing delimiters may be elided, with multiple elisions then representing null strings, but the first delimiter must always be present. In any delimited text, newline may not appear literally; `\n` may be typed for newline; and `\/` quotes the delimiter, here `/`. Backslash is other interpreted literally, except in **s** commands.

Most commands may be prefixed with an address to indicate their range of operation. Those that may not are marked with a * below. If a command takes an address and none is supplised, dot is used. The sole exception is the **w** command, which defaults to `0,$`. In the description, “range” is used to represent whatever address is supplied. Many commands set the value of dot as a side effect. If so, it is always to the “result” of the change: the empty string for a deletion, the new text for an insertion, etc. (but see the **s** and **e** commands) .

Text commands

a/text/ Insert the text into the file after the range. Set dot.

May also be written as

```
a
lines
of
text
.
```

c or **i** Same as **a**, but **c** replaces the text, while **i** inserts *before* the range.

d Delete the text in range. Set dot.

s/regexp/text/
Substitute **text** for the first match to the regular expression in the range. Set dot to the modified range. In**text** the character & stands for the string that matched the e xpression. Backslash behaves as usual unless followed by a digit: **\d** stands for the string that matched the subexpression begun by the **d**-th left parenthesis. If **s** is followed immediately by a number **n**, as in `s2/x/y/`, the **n**-th match in the range is substituted. If the command is followed by **g**, as in `s/x/y/g`, all matches in the range are substituted.

m a1 Move the range to after **a1**. Set dot.

t a1 Copy the range to after **a1**. Set dot.

Display commands

p Print the text in the range. Set dot.

= Print the line address and character address of the range.

=# Print just the character address of the range.

File commands.

∗ **b file-list**
Set the current file to the first file named in the list that `sam` also has in its menu. The list may be expressed **<shell-command** in which case the file names are taken as words (in the shell sense) generated by the shell command.

∗ **B file-list**
Same as **b**, except that filenames not in the menu are entered there, and all file names in the list are examined.

∗ **n** Print a menu of files. The format is:

’ or blank
indicating the file is modified or clean,

- or + indicating the file is unread or has been read (in the terminal, ∗ means more than one window is open),

. or blank
indicating the current file,

a blank, and the filename.

∗ **D file-list**
Delete the named files from the menu. If no files are named, the current file is deleted. It is an error to delete a modified file, but a subsequent **D** will delete such a file.

I/O commands

∗ **e filename**
Replace the file by the contents of the named external file. Set dot to the beginning of the file.

r filename
Replace the text in the range by the contents of the named external file. Set dot.

w filename
Write the range (default `0,$`) to the named external file.

∗ **f filename**
Set the file name and print the resulting menu entry.

If the file name argument is absent from any of these, the current file name is used. **e** al ways sets the file name, **r** and **w** will do so if the file has no name.

< shell-command
Replace the range by the standard output of the shell command.

> shell-command
Sends the range to the standard input of the shell command.

| shell-command
: Send the range to the standard input, and replace it by the standard output, of the shell command.

∗ **! shell-command**
: Run the shell command.

∗ **cd directory**
: Change working directory. If no directory is specified,`$HOME` is used.

In any of <, >, | **, or** !, if the shell command is omitted, the last shell command (of any type) is substituted. If **`sam`** is downloaded, ! sets standard input to `/dev/null`, and otherwise unassigned output (`stdout` for ! and>, `stderr` for all) is placed in `${HOME}/sam.err` and the first fe w lines are printed.

Loops and conditionals

x/regexp/ command
: For each match of the regular expression in the range, run the command with dot set to the match. Set dot to the last match. If the regular expression and its slashes are omitted, `/.*\n/` is assumed. Null string matches potentially occur before every character of the range and at the end of the range.

y/regexp/ command
: Like **x**, but run the command for each substring that lies before, between, or after the matches that would be generated by **x**. There is no default behavior. Null substrings potentially occur before every character in the range.

∗ **X/regexp/ command**
: For each file whose menu entry matches the regular expression, make that the current file and run the command. If the expression is omitted, the command is run in every file.

∗ **Y/regexp/ command**
: Same as **X**, but for files that do not match the regular expression, and the expression is required.

g/regexp/ command

v/regexp/ command
: If the range contains (**g**) or does not contain (**v**) a match for the e xpression, set dot to the range and run the command.

These may be nested arbitrarily deeply, but only one instance of either **X** or **Y** may appear in a single command. An empty command in an**x** or **y** def aults to **p**; an empty command in **X** or **Y** defaults to **f**. **g** and **v** do not have defaults.

Miscellany

k Set the current file's mark to the range. Does not set dot.

∗ **q** Quit. It is an error to quit with modified files, but a second **q** will succeed.

∗ **u n** Undo the last **n** (default 1) top-level commands that changed the contents or name of the current file, and any other file whose most recent change was simultaneous with the current file's change. Successive **u** commands move further back in time. The only commands for which **u** is ineffective are **cd**, **u**, **q**, **w**, and **D**.

empty If the range is explicit, set dot to the range. If **`sam`** is downloaded, the resulting dot is selected on the screen; otherwise it is printed. If no address is specified (the command is a newline) dot is extended in either direction to the line boundaries and printed. If dot is thereby unchanges, i is set to `.+1` and printed.

Grouping and multiple changes

Commands may be groups by enclosing them in curly braces. Commands within the braces must appear on separate lines (no backslashes are required between commands). Semantically, the opening brance is like a command: it takes an (optional) address and sets dot for each sub-command. Commands within the braces are executed sequentially, but changes made by one command are not visible to other commands (see the next paragraph). Braces may be nested arbitrarily.

When a command makes a number of changes to a file, as in `x/re/c/text/`, the addresses of all changes to the file are computed in the original file. If the changes are in sequence, they are applied to the file. Successive insertions at the same address are catenated into a single insertion composed of the several insertions in the order applied.

The terminal

What follows refers to the behavior of **`sam`** when downloaded, that is, when operating as a display editor on a bitmap display. This is the default behavior; invoking **`sam`** with the **`−d`** (no download) option provides access to the command language only.

Each file may have zero or more windows open. Each window is equivalent and is updated simultaneously with changes in other windows on the same file. Each window has an independent value of dot, indicated by a highlighted substring on the display. Dot may be in a region not within the window. There is usually a “current window”, marked with a dark border, to which typed text and editing commands apply. The escape key selects (sets dot to) text typed since the last mouse button hit.

The button 3 menu controls window operations. The top of the menu provides the following operators, each of which uses one or more cursors to prompt for selection of a window or sweeping of a rectangle.

new Create a new empty file: Depress button 3 where one corner of the new rectangle should appear (box cursor), and move the mouse while holding down button 3 to the diagonally opposite corner. “Sweeping” a null rectangle gets a large window disjoint from the command window or the whole **`sam`** window, depending on where the null rectangle is.

zerox Create a copy of an existing window. After selecting the window to copy with button 1, sweep out the window for the copy.

reshape
Change the size and location of a window. First click button 3 in the window to be changed (gunsight cursor). Then sweep out a window as for the **new** menu selection.

close Delete the window. In the last window of a file, **close** is equivalent to a **D** for the file.

write Equivalent to a **w** for the file.

Below these operators is a list of available files, starting with **˜˜sam˜˜**, the command window. Selecting a file from the list makes the most recently used window on that file current, unless it is already current, in which case selections cycle through the open windows. If no windows are open on the file, the user is prompted to open one. Files other than **˜˜sam˜˜** are marked with one of the characters `-+*` according as zero, one, or more windows are open on the file. A further mark, ., appears on the file in the current window and a single quote, `'`, on a file modified since last write.

The command window, created automatically when **`sam`** starts, is an ordinary window except that text typed to it is interpreted as commands for the editor rather than passive text, and text printed by editor commands appears in it. There is an “output point” that separates commands being typed from previous output. Commands typed in the command window apply to the current open file–the file in the most recently current window.

Manipulating text

Typed characters replace the current selection (dot) in the current window. Backspace deletes the previous character. Escape selects (sets dot to) everything typed since the last mouse hit.

Button 1 changes selection. Pointing to a non-current window with button 1 makes it current; within the current window, button 1 selects text, thus setting dot. Double-clicking selects text to the boundaries of words, lines, quoted strings, or bracketed strings, depending on the text at the click.

Button 2 provides a menu of editing commands:

cut Delete dot and save the delted text in the snarf buffer.

paste Replace the text in dot by the contents of the snarf buffer.

snarf Save the text in dot in the snarf buffer.

look Search forward for the next occurence of the literal text in dot. If dot is the null string, the text in the snarf buffer is used. The snarf buffer is unaffected.

<exch>
Exchange the snarf buffer with the current system-wide text selection. The exchange of a large amount of selected text is truncated to the size of the editor's internal snarf buffer (currently 4K) without warning.

/regexp
Search forward for the next match of the last regular expression typed in a command. (Not in command window.)

send Send the text in dot, or the snarf buffer if dot is the null string, as if it were typed to the command window. Saves the sent text in the snarf buffer. (Command window only.)

Abnormal termination

If **`sam`** terminates other than by a **q** command (by hangup, deleting its window, etc.), modified files are saved in an executable file, (${HOME}/sam.save). This program, when executed, asks whether to write each file back to an external file. The answer **y** causes writing; anything else skips the file. If a machine crash prevents the creation of a file, all changes are lost. If an editing session is difficult to replicate, writing changed files often is recommended.

Remote execution

`sam` allows the host and terminal parts of the editor to run on diffrent machines, in a process called "downloading". This process can be suppressed with the**`−d`** option, which then runs only the host part in the manner of `ed`(1).

Running the host part on another machine is accomplished using the **`−r`** option, which is used to specify a remote machine name suitable for passing to the remote shell command specified in the `RSH` environment variable.

By default, **`sam`** will run a command called **`rsam`** as the host-part on the remote machine. **`rsam`** opens up an additional control channel on the remote machine, allowing **`sam`** to be controlled via the **`B`** command on the remote machine as well.

The only components of **`sam`** that need to be on the remote machine are **`rsam`** and **`sam`**, and any command specified as the argument to the **`−s`** option. Users may also lik e to have the **`B`** command present on the remote system; invoking this command on the remote system will (if **`sam`** was invoked with its default remote host command, i.e. **`rsam`**) open files in the local terminal. This allo ws users to run the terminal part of **`sam`** locally while controlling it via a remote shell connection.

Controlling running instances of `sam`

`B` is a shell command that causes a downloaded instance of **`sam`** to load the named files. The **`-r`** option causes the instance of **`sam`** connected to *`machine`* to load the named files; the default is the most-recently started local instance.

`B` may also be called on a remote machine, causing the downloaded instance of sam connected to that machine to load the named files.

Unicode Text Input

`sam` allows the input of arbitrary Unicode characters from the Basic Multilingual Plane (BMP) via five-character and two-character sequences. These sequences are entered while holding down the system compose key (on most keyboards, this key is labeled **Alt**).

The first method allows the entry of any code point in the BMP. While holding down the compose key, an uppercase `X` character is typed, followed by exactly four lowercase hexadecimal digits naming the codepoint.

Commonly used codepoints can be entered with an abbreviated two-character sequence. These sequence definitions are read from a file called `.keyboard` in the user's home directory. Each line in this file consists of two characters, followed by any number of spaces, and a hexadecimal number. Holding down the compose key and typing the two listed characters will insert the codepoint indicated by the number. For example, given the `.keyboard` line:

```
12 0x00BD
```

then typing the characters 1 and 2 while holding down the compose key will insert Unicode code point 0x00BD (½) into the file.

After the hexadecimal codepoint specification, the rest of the line is ignored and is therefore usable as a comment field.

If no `.keyboard` file is present, the following key sequences are defined by default:

Keys	Codepoint	Keys	Codepoint	Keys	Codepoint	Keys	Codepoint
!!	¡	c$	¢	l$	£	g$	¤
y$	¥	||	¦	SS	§	""	¨
cO	©	sa	ª	<<	«	no	¬
--		rO	®	__	¯	de	°
+-	±	s2	²	s3	³	’’	´
mi	µ	pg	¶	..	·	,,	¸
s1	¹	so	º	>>	»	14	¼
12	½	34	¾	??	¿	‘A	À
’A	Á	ˆA	Â	˜A	Ã	"A	Ä
oA	Å	AE	Æ	,C	Ç	‘E	È
’E	É	ˆE	Ê	"E	Ë	‘I	Ì
’I	Í	ˆI	Î	"I	Ï	D-	Ð
˜N	Ñ	‘O	Ò	’O	Ó	ˆO	Ô
˜O	Õ	"O	Ö	mu	×	/O	Ø
‘U	Ù	’U	Ú	ˆU	Û	"U	Ü
’Y	Ý	|P	Þ	ss	ß	‘a	à
’a	á	ˆa	â	˜a	ã	"a	ä
oa	å	ae	æ	,c	ç	‘e	è
’e	é	ˆe	ê	"e	ë	‘i	ì
’i	í	ˆi	î	"i	ï	d-	ð
˜n	ñ	‘o	ò	’o	ó	ˆo	ô
˜o	õ	"o	ö	-:	÷	/o	ø
‘u	ù	’u	ú	ˆu	û	"u	ü
’y	ý	|p	þ	"y	ÿ	wk	

Keys	Codepoint	Keys	Codepoint	Keys	Codepoint	Keys	Codepoint
wq		wr		wb		wn	
wp		bk		bq		br	
bb		bn		bp		*a	α
*b	β	*g	γ	*d	δ	*e	ε
*z	ζ	*y	η	*h	θ	*i	ι
*k	κ	*l	λ	*m	μ	*n	ν
*c	ξ	*o	ο	*p	π	*r	ρ
ts	ς	*s	σ	*t	τ	*u	υ
*f	φ	*x	χ	*q	ψ	*w	ω
*A	Α	*B	Β	*G	Γ	*D	Δ
*E	Ε	*Z	Ζ	*Y	Η	*H	Θ
*I	Ι	*K	Κ	*L	Λ	*M	Μ
*N	Ν	*C	Ξ	*O	Ο	*P	Π
*R	Ρ	*S	Σ	*T	Τ	*U	ϒ
*F	Φ	*X	Χ	*Q	Ψ	*W	Ω
<-	←	ua	↑	->	→	da	↓
ab	↔	V=	⇐	=V	⇒	fa	∀
te	∃	pd	∂	es	∅	De	Δ
gr	∇	mo	∈	!m	∉	st	
**	∗	bu		sr	√	pt	∝
if	∞	an	∠	l&	∧	l|	∨
ca	∩	cu	∪	is	∫	tf	∴
˜=	≃	cg	≅	˜˜	≈	!=	≠
==	≡	<=		>=		sb	⊂
sp	⊃	!b	⊄	ib	⊆	ip	⊇
O+	⊕	O-		Ox	⊗	tu	
Tu		lz		el		:(	
:)		;)					

ENVIRONMENT

The following environment variables affect the operation of **`sam`**:

`FOREGROUND`
: Sets the foreground color used by **`sam`** to draw its terminal. Common English color names can be used (see `rgb`(5)), or e xact colors can be specified as **#rrggbb**, where **rr**, **gg**, and **bb** are hexadecimal digits describing the red, green, and blue components of the color, respectively. By default, this is the string “black”.

`BACKGROUND`
: As `FOREGROUND`, but describing the background color used to draw the terminal. By default, this is the string “white”.

`FONT` A string representing a `fc-match`(1) compatible font pattern. The font described by this pattern will be used to render text in the terminal. By default, this is the string “monospace”.

`RSH` The name of a command to be used to connect to a remote machine when **`sam`** is invoked with the **`-r`** option. It will be passed at least two arguments: the name of the machine to connect to and the name of the remote command to execute (e.g. **`rsam`**). Any additional arguments should be passed to the command on the remote machine. By default, this is the string “ssh”.

FILES

`${HOME}/.keyboard`
: Provides a mapping of two-character sequences to Unicode code points. Note that the code points must be in the Basic Multilingual Plane.

`${HOME}/sam.save`
: Created if **`sam`** terminates abnormally. Executing this file will prompt the user to restore the files that were being edited at the time of termination.

`${HOME}/sam.err`
: Stores output of shell commands executed by **`sam`**.

SEE ALSO

`ed`(1)

BUGS

When a **`sam`** window is resized, the command window may have the wrong size.

Under some window managers, resizing the window may cause a panic.

`sam` has issues with compositing window managers like compiz, resulting in some rendering glitches.

The only human language in which colors may be specified is English.

The only human language in which output is generated is English.

There is no support for right-to-left text, ligatures, composed characters, or any other complex text rendering.