Rocky Enterprise Linux 9.2 Manual Pages on command 'history.3readline'

C:\>man history.3readline

HISTORY(3) Library Functions Manual HISTORY(3)

NAME

history - GNU History Library

COPYRIGHT

The GNU History Library is Copyright (C) 1989-2017 by the Free Software Foundation, Inc.

DESCRIPTION

Many programs read input from the user a line at a time. The GNU History library is able to keep track of those lines, associate arbitrary data with each line, and utilize information from previous lines in composing new ones.

HISTORY EXPANSION

The history library supports a history expansion feature that is identical to the history expansion in bash. This section describes what syntax features are avail? able.

History expansions introduce words from the history list into the input stream, making it easy to repeat commands, insert the arguments to a previous command into the current input line, or fix errors in previous commands quickly.

History expansion is usually performed immediately after a complete line is read. It takes place in two parts. The first is to determine which line from the history list to use during substitution. The second is to select portions of that line for inclusion into the current one. The line selected from the history is the event, and the portions of that line that are acted upon are words. Various modifiers are

available to manipulate the selected words. The line is broken into words in the same fashion as bash does when reading input, so that several words that would oth? erwise be separated are considered one word when surrounded by quotes (see the de? scription of history_tokenize() below). History expansions are introduced by the appearance of the history expansion character, which is ! by default. Only back? slash (\) and single quotes can quote the history expansion character.

Event Designators

An event designator is a reference to a command line entry in the history list. Unless the reference is absolute, events are relative to the current position in the history list.

! Start a history substitution, except when followed by a blank, newline, = or (.

!n Refer to command line n.

!-n Refer to the current command minus n.

!! Refer to the previous command. This is a synonym for `!-1'.

!string

Refer to the most recent command preceding the current position in the his? tory list starting with string.

!?string[?]

Refer to the most recent command preceding the current position in the his? tory list containing string. The trailing ? may be omitted if string is followed immediately by a newline.

^string1^string2^

Quick substitution. Repeat the last command, replacing string1 with string2. Equivalent to ``!!:s/string1/string2/'' (see Modifiers below).

!# The entire command line typed so far.

Word Designators

Word designators are used to select desired words from the event. A : separates the event specification from the word designator. It may be omitted if the word designator begins with a ^, $, *, -, or %. Words are numbered from the beginning of the line, with the first word being denoted by 0 (zero). Words are inserted into the current line separated by single spaces.

0 (zero)

The zeroth word. For the shell, this is the command word.

n The nth word.

^ The first argument. That is, word 1.

$ The last word. This is usually the last argument, but will expand to the zeroth word if there is only one word in the line.

% The word matched by the most recent `?string?' search.

x-y A range of words; `-y' abbreviates `0-y'.

* All of the words but the zeroth. This is a synonym for `1-$'. It is not an error to use * if there is just one word in the event; the empty string is returned in that case.

x* Abbreviates x-$.

x- Abbreviates x-$ like x*, but omits the last word.

If a word designator is supplied without an event specification, the previous com? mand is used as the event.

Modifiers

After the optional word designator, there may appear a sequence of one or more of the following modifiers, each preceded by a `:'.

h Remove a trailing file name component, leaving only the head.

t Remove all leading file name components, leaving the tail.

r Remove a trailing suffix of the form .xxx, leaving the basename.

e Remove all but the trailing suffix.

p Print the new command but do not execute it.

q Quote the substituted words, escaping further substitutions.

x Quote the substituted words as with q, but break into words at blanks and newlines.

s/old/new/

Substitute new for the first occurrence of old in the event line. Any de? limiter can be used in place of /. The final delimiter is optional if it is the last character of the event line. The delimiter may be quoted in old and new with a single backslash. If & appears in new, it is replaced by old. A single backslash will quote the &. If old is null, it is set to the last old substituted, or, if no previous history substitutions took place, the last string in a !?string[?] search.

& Repeat the previous substitution.

g Cause changes to be applied over the entire event line. This is used in conjunction with `:s' (e.g., `:gs/old/new/') or `:&'. If used with `:s', any delimiter can be used in place of /, and the final delimiter is optional if it is the last character of the event line. An a may be used as a syn? onym for g.

G Apply the following `s' modifier once to each word in the event line.

PROGRAMMING WITH HISTORY FUNCTIONS

This section describes how to use the History library in other programs.

Introduction to History

The programmer using the History library has available functions for remembering lines on a history list, associating arbitrary data with a line, removing lines from the list, searching through the list for a line containing an arbitrary text string, and referencing any line in the list directly. In addition, a history ex? pansion function is available which provides for a consistent user interface across different programs.

The user using programs written with the History library has the benefit of a con? sistent user interface with a set of well-known commands for manipulating the text of previous lines and using that text in new commands. The basic history manipula? tion commands are identical to the history substitution provided by bash.

If the programmer desires, he can use the Readline library, which includes some history manipulation by default, and has the added advantage of command line edit? ing.

Before declaring any functions using any functionality the History library provides in other code, an application writer should include the file <readline/history.h> in any file that uses the History library's features. It supplies extern declara? tions for all of the library's public functions and variables, and declares all of the public data structures.

History Storage

The history list is an array of history entries. A history entry is declared as follows:

```
typedef void * histdata_t;

typedef struct _hist_entry {
```

```
  char *line;
  char *timestamp;
  histdata_t data;
} HIST_ENTRY;
```

The history list itself might therefore be declared as

```
HIST_ENTRY ** the_history_list;
```

The state of the History library is encapsulated into a single structure:

```
/*
 * A structure used to pass around the current state of the history.
 */
typedef struct _hist_state {
  HIST_ENTRY **entries; /* Pointer to the entries themselves. */
  int offset;          /* The location pointer within this array. */
  int length;          /* Number of elements within this array. */
  int size;            /* Number of slots allocated to this array. */
  int flags;
} HISTORY_STATE;
```

If the flags member includes HS_STIFLED, the history has been stifled.

History Functions

This section describes the calling sequence for the various functions exported by the GNU History library.

Initializing History and State Management

This section describes functions used to initialize and manage the state of the History library when you want to use the history functions in your program.

```
void using_history (void)
```

Begin a session in which the history functions might be used. This initializes the interactive variables.

```
HISTORY_STATE * history_get_history_state (void)
```

Return a structure describing the current state of the input history.

```
void history_set_history_state (HISTORY_STATE *state)
```

Set the state of the history list according to state.

History List Management

These functions manage individual entries on the history list, or set parameters

managing the list itself.

void add_history (const char *string)

Place string at the end of the history list. The associated data field (if any) is set to NULL. If the maximum number of history entries has been set using sti? fle_history(), and the new number of history entries would exceed that maximum, the oldest history entry is removed.

void add_history_time (const char *string)

Change the time stamp associated with the most recent history entry to string.

HIST_ENTRY * remove_history (int which)

Remove history entry at offset which from the history. The removed element is re? turned so you can free the line, data, and containing structure.

histdata_t free_history_entry (HIST_ENTRY *histent)

Free the history entry histent and any history library private data associated with it. Returns the application-specific data so the caller can dispose of it.

HIST_ENTRY * replace_history_entry (int which, const char *line, histdata_t data)

Make the history entry at offset which have line and data. This returns the old entry so the caller can dispose of any application-specific data. In the case of an invalid which, a NULL pointer is returned.

void clear_history (void)

Clear the history list by deleting all the entries.

void stifle_history (int max)

Stifle the history list, remembering only the last max entries. The history list will contain only max entries at a time.

int unstifle_history (void)

Stop stifling the history. This returns the previously-set maximum number of his? tory entries (as set by stifle_history()). history was stifled. The value is pos? itive if the history was stifled, negative if it wasn't.

int history_is_stifled (void)

Returns non-zero if the history is stifled, zero if it is not.

Information About the History List

These functions return information about the entire history list or individual list entries.

HIST_ENTRY ** history_list (void)

Return a NULL terminated array of HIST_ENTRY * which is the current input history. Element 0 of this list is the beginning of time. If there is no history, return NULL.

int where_history (void)

Returns the offset of the current history element.

HIST_ENTRY * current_history (void)

Return the history entry at the current position, as determined by where_history(). If there is no entry there, return a NULL pointer.

HIST_ENTRY * history_get (int offset)

Return the history entry at position offset. The range of valid values of offset starts at history_base and ends at history_length - 1. If there is no entry there, or if offset is outside the valid range, return a NULL pointer.

time_t history_get_time (HIST_ENTRY *)

Return the time stamp associated with the history entry passed as the argument.

int history_total_bytes (void)

Return the number of bytes that the primary history entries are using. This func? tion returns the sum of the lengths of all the lines in the history.

Moving Around the History List

These functions allow the current index into the history list to be set or changed.

int history_set_pos (int pos)

Set the current history offset to pos, an absolute index into the list. Returns 1 on success, 0 if pos is less than zero or greater than the number of history en? tries.

HIST_ENTRY * previous_history (void)

Back up the current history offset to the previous history entry, and return a pointer to that entry. If there is no previous entry, return a NULL pointer.

HIST_ENTRY * next_history (void)

If the current history offset refers to a valid history entry, increment the cur? rent history offset. If the possibly-incremented history offset refers to a valid history entry, return a pointer to that entry; otherwise, return a NULL pointer.

Searching the History List

These functions allow searching of the history list for entries containing a spe? cific string. Searching may be performed both forward and backward from the cur?

rent history position. The search may be anchored, meaning that the string must match at the beginning of the history entry.

int history_search (const char *string, int direction)

Search the history for string, starting at the current history offset. If direc? tion is less than 0, then the search is through previous entries, otherwise through subsequent entries. If string is found, then the current history index is set to that history entry, and the value returned is the offset in the line of the entry where string was found. Otherwise, nothing is changed, and a -1 is returned.

int history_search_prefix (const char *string, int direction)

Search the history for string, starting at the current history offset. The search is anchored: matching lines must begin with string. If direction is less than 0, then the search is through previous entries, otherwise through subsequent entries. If string is found, then the current history index is set to that entry, and the return value is 0. Otherwise, nothing is changed, and a -1 is returned.

int history_search_pos (const char *string, int direction, int pos)

Search for string in the history list, starting at pos, an absolute index into the list. If direction is negative, the search proceeds backward from pos, otherwise forward. Returns the absolute index of the history element where string was found, or -1 otherwise.

Managing the History File

The History library can read the history from and write it to a file. This section documents the functions for managing a history file.

int read_history (const char *filename)

Add the contents of filename to the history list, a line at a time. If filename is NULL, then read from ~/.history. Returns 0 if successful, or errno if not.

int read_history_range (const char *filename, int from, int to)

Read a range of lines from filename, adding them to the history list. Start read? ing at line from and end at to. If from is zero, start at the beginning. If to is less than from, then read until the end of the file. If filename is NULL, then read from ~/.history. Returns 0 if successful, or errno if not.

int write_history (const char *filename)

Write the current history to filename, overwriting filename if necessary. If file? name is NULL, then write the history list to ~/.history. Returns 0 on success, or

errno on a read or write error.

int append_history (int nelements, const char *filename)

Append the last nelements of the history list to filename. If filename is NULL, then append to ~/.history. Returns 0 on success, or errno on a read or write er? ror.

int history_truncate_file (const char *filename, int nlines)

Truncate the history file filename, leaving only the last nlines lines. If file? name is NULL, then ~/.history is truncated. Returns 0 on success, or errno on failure.

History Expansion

These functions implement history expansion.

int history_expand (char *string, char **output)

Expand string, placing the result into output, a pointer to a string. Returns:

- 0 If no expansions took place (or, if the only change in the text was the removal of escape characters preceding the history expansion character);
- 1 if expansions did take place;
- -1 if there was an error in expansion;
- 2 if the returned line should be displayed, but not executed, as with the :p modifier.

If an error ocurred in expansion, then output contains a descriptive error message.

char * get_history_event (const char *string, int *cindex, int qchar)

Returns the text of the history event beginning at string + *cindex. *cindex is modified to point to after the event specifier. At function entry, cindex points to the index into string where the history event specification begins. qchar is a character that is allowed to end the event specification in addition to the ``nor? mal'' terminating characters.

char ** history_tokenize (const char *string)

Return an array of tokens parsed out of string, much as the shell might. The to? kens are split on the characters in the history_word_delimiters variable, and shell quoting conventions are obeyed.

char * history_arg_extract (int first, int last, const char *string)

Extract a string segment consisting of the first through last arguments present in

string. Arguments are split using history_tokenize().

History Variables

This section describes the externally-visible variables exported by the GNU History Library.

int history_base

The logical offset of the first entry in the history list.

int history_length

The number of entries currently stored in the history list.

int history_max_entries

The maximum number of history entries. This must be changed using stifle_history().

int history_wite_timestamps

If non-zero, timestamps are written to the history file, so they can be preserved between sessions. The default value is 0, meaning that timestamps are not saved. The current timestamp format uses the value of history_comment_char to delimit timestamp entries in the history file. If that variable does not have a value (the default), timestamps will not be written.

char history_expansion_char

The character that introduces a history event. The default is !. Setting this to 0 inhibits history expansion.

char history_subst_char

The character that invokes word substitution if found at the start of a line. The default is ^.

char history_comment_char

During tokenization, if this character is seen as the first character of a word, then it and all subsequent characters up to a newline are ignored, suppressing history expansion for the remainder of the line. This is disabled by default.

char * history_word_delimiters

The characters that separate tokens for history_tokenize(). The default value is " \t\n()<>;&|".

char * history_no_expand_chars

The list of characters which inhibit history expansion if found immediately following history_expansion_char. The default is space, tab, newline, \r, and =.

char * history_search_delimiter_chars

The list of additional characters which can delimit a history search string, in ad?

dition to space, tab, : and ? in the case of a substring search. The default is

empty.

int history_quotes_inhibit_expansion

If non-zero, double-quoted words are not scanned for the history expansion charac?

ter or the history comment character. The default value is 0.

rl_linebuf_func_t * history_inhibit_expansion_function

This should be set to the address of a function that takes two arguments: a char *

(string) and an int index into that string (i). It should return a non-zero value

if the history expansion starting at string[i] should not be performed; zero if the

expansion should be done. It is intended for use by applications like bash that

use the history expansion character for additional purposes. By default, this

variable is set to NULL.

FILES

~/.history

Default filename for reading and writing saved history

SEE ALSO

The Gnu Readline Library, Brian Fox and Chet Ramey

The Gnu History Library, Brian Fox and Chet Ramey

bash(1)

readline(3)

AUTHORS

Brian Fox, Free Software Foundation

bfox@gnu.org

Chet Ramey, Case Western Reserve University

chet.ramey@case.edu

BUG REPORTS

If you find a bug in the history library, you should report it. But first, you

should make sure that it really is a bug, and that it appears in the latest version

of the history library that you have.

Once you have determined that a bug actually exists, mail a bug report to bug-read?

line@gnu.org. If you have a fix, you are welcome to mail that as well! Sugges?

tions and `philosophical' bug reports may be mailed to bug-readline@gnu.org or posted to the Usenet newsgroup gnu.bash.bug.

Comments and bug reports concerning this manual page should be directed to chet.ramey@case.edu.