GNU History Library

March 1996

Brian Fox, Free Software Foundation
Chet Ramey, Case Western Reserve University
This document describes the GNU History library, a programming tool that provides a consistent user interface for recalling lines of previously typed input.

Published by the Free Software Foundation
675 Massachusetts Avenue,
Cambridge, MA 02139 USA

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Foundation.

Copyright © 1989, 1991 Free Software Foundation, Inc.
1 Using History Interactively

This chapter describes how to use the GNU History Library interactively, from a user's standpoint. It should be considered a user's guide. For information on using the GNU History Library in your own programs, see Chapter 2 [Programming with GNU History], page 3.

1.1 Interactive History Expansion

The History library provides a history expansion feature that is similar to the history expansion provided by csh. This section describes the syntax used to manipulate the history information.

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 takes place in two parts. The first is to determine which line from the previous history should be used during substitution. The second is to select portions of that line for inclusion into the current one. The line selected from the previous history is called the event, and the portions of that line that are acted upon are called words. Various modifiers are available to manipulate the selected words. The line is broken into words in the same fashion that Bash does, so that several English (or Unix) words surrounded by quotes are considered as one word. History expansions are introduced by the appearance of the history expansion character, which is ‘!’ by default.

1.1.1 Event Designators

An event designator is a reference to a command line entry in the history list.

! Start a history substitution, except when followed by a space, tab, the end of the line, \ or \.

!n Refer to command line n.

!-n Refer to the command n lines back.

!! Refer to the previous command. This is a synonym for ‘!-1’.

!string Refer to the most recent command starting with string.

!?string Refer to the most recent command containing string. The trailing ‘?’ may be omitted if the string is followed immediately by a newline.

"string1"string2" Quick Substitution. Repeat the last command, replacing string1 with string2. Equivalent to !! :s/string1/string2/.

!# The entire command line typed so far.
1.1.2 Word Designators

Word designators are used to select desired words from the event. A ‘:’ separates the event specification from the word designator. It can 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 0th word. For many applications, this is the command word.
n The nth word.
~ The first argument; that is, word 1.
$ The last argument.
% The word matched by the most recent ‘?string?’ search.
x-y A range of words; ‘-y’ abbreviates ‘0-y’.
* All of the words, except the 0th. 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 command is used as the event.

1.1.3 Modifiers

After the optional word designator, you can add a sequence of one or more of the following modifiers, each preceded by a ‘:’.

h Remove a trailing pathname component, leaving only the head.
t Remove all leading pathname components, leaving the tail.
r Remove a trailing suffix of the form ‘.suffix’, leaving the basename.
e Remove all but the trailing suffix.
p Print the new command but do not execute it.

s/old/new/ Substitute new for the first occurrence of old in the event line. Any delimiter may be used in place of ‘/’. 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 ‘&’. The final delimiter is optional if it is the last character on the input line.

& Repeat the previous substitution.

g Cause changes to be applied over the entire event line. Used in conjunction with ‘s’, as in gs/old/new/, or with ‘&’. 
2 Programming with GNU History

This chapter describes how to interface programs that you write with the GNU History Library. It should be considered a technical guide. For information on the interactive use of GNU History, see Chapter 1 [Using History Interactively], page 1.

2.1 Introduction to History

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.

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 expansion 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 consistent 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 manipulation commands are similar to the history substitution provided by csh.

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 editing.

2.2 History Storage

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

```c
typedef struct _hist_entry {
    char *line;
    char *data;
} HIST_ENTRY;
```

The history list itself might therefore be declared as

```c
HIST_ENTRY **the_history_list;
```

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

```c
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.
2.3 History Functions

This section describes the calling sequence for the various functions present in GNU History.

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

HISTORY_STATE * history_get_history_state () Function
Return a structure describing the current state of the input history.

void history_set_history_state (HISTORY_STATE *state) Function
Set the state of the history list according to state.

2.3.2 History List Management

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

void add_history (char *string) Function
Place string at the end of the history list. The associated data field (if any) is set to NULL.

HIST_ENTRY * remove_history (int which) Function
Remove history entry at offset which from the history. The removed element is returned so you can free the line, data, and containing structure.

HIST_ENTRY * replace_history_entry (int which, char *line, char *data) Function
Make the history entry at offset which have line and data. This returns the old entry so you can dispose of the data. In the case of an invalid which, a NULL pointer is returned.

void clear_history () Function
Clear the history list by deleting all the entries.

void stifle_history (int max) Function
Stifle the history list, remembering only the last max entries.

int unstifle_history () Function
Stop stifling the history. This returns the previous amount the history was stifled. The value is positive if the history was stifled, negative if it wasn't.

int history_is_stifled () Function
Returns non-zero if the history is stifled, zero if it is not.
2.3.3 Information About the History List

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

**Function**

HIST_ENTRY ** history_list ()

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.

**Function**

int where_history ()

Returns the offset of the current history element.

**Function**

HIST_ENTRY * current_history ()

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

**Function**

HIST_ENTRY * history_get (int offset)

Return the history entry at position offset, starting from history_base. If there is no entry there, or if offset is greater than the history length, return a NULL pointer.

**Function**

int history_total_bytes ()

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

2.3.4 Moving Around the History List

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

**Function**

int history_set_pos (int pos)

Set the position in the history list to pos, an absolute index into the list.

**Function**

HIST_ENTRY * previous_history ()

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.

**Function**

HIST_ENTRY * next_history ()

Move the current history offset forward to the next history entry, and return the a pointer to that entry. If there is no next entry, return a NULL pointer.

2.3.5 Searching the History List

These functions allow searching of the history list for entries containing a specific string. Searching may be performed both forward and backward from the current history position. The search may be anchored, meaning that the string must match at the beginning of the history entry.

**Function**

int history_search (char *string, int direction)

Search the history for string, starting at the current history offset. If direction < 0, then the search is through previous entries, else through subsequent. 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.

```c
int history_search_prefix (char *string, int direction) Function
Search the history for string, starting at the current history offset. The search
is anchored: matching lines must begin with string. If direction < 0, then the
search is through previous entries, else through subsequent. 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.
```

```c
int history_search_pos (char *string, int direction, int pos) Function
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.
```

### 2.3.6 Managing the History File

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

```c
int read_history (char *filename) Function
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.
```

```c
int read_history_range (char *filename, int from, int to) Function
Read a range of lines from filename, adding them to the history list. Start
reading 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.
```

```c
int write_history (char *filename) Function
Write the current history to filename, overwriting filename if necessary. If
filename is NULL, then write the history list to `~/.history'. Values returned
are as in read_history ()..
```

```c
int append_history (int nelements, char *filename) Function
Append the last nelements of the history list to filename.
```

```c
int history_truncate_file (char *filename, int nlines) Function
Truncate the history file filename, leaving only the last nlines lines.
```

### 2.3.7 History Expansion

These functions implement csh-like history expansion.
int history_expand (char *string, char **output)  
Function
Expand string, placing the result into output, a pointer to a string (see Section 1.1 [History Interaction], page 1). Returns:

0  If no expansions took place (or, if the only change in the text was the deslashifying of the history expansion character);
1  if expansions did take place;
-1 if there was an error in expansion;
2  if the returned line should only be displayed, but not executed, as with the :p modifier (see Section 1.1.3 [Modifiers], page 2).

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

char * history_arg_extract (int first, int last, char *string)  
Function
Extract a string segment consisting of the first through last arguments present in string. Arguments are broken up as in Bash.

char * get_history_event (char *string, int *cindex, int qchar)  
Function
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 “normal” terminating characters.

char ** history_tokenize (char *string)  
Function
Return an array of tokens parsed out of string, much as the shell might. The tokens are split on white space and on the characters ()<>;:&\$, and shell quoting conventions are obeyed.

2.4 History Variables

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

int history_base  
Variable
The logical offset of the first entry in the history list.

int history_length  
Variable
The number of entries currently stored in the history list.

int max_input_history  
Variable
The maximum number of history entries. This must be changed using stifle_history.

char history_expansion_char  
Variable
The character that starts a history event. The default is ‘!’.
char history_subst_char Variable
The character that invokes word substitution if found at the start of a line. The
default is ‘‘.”

char history_comment_char Variable
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_no_expand_chars Variable
The list of characters which inhibit history expansion if found immediately
following history_expansion_char. The default is whitespace and ‘=’.

char * history_search_delimiter_chars Variable
The list of additional characters which can delimit a history search string, in
addition to whitespace, ‘:’ and ‘?’ in the case of a substring search. The default
is empty.

int history_quotes_inhibit_expansion Variable
If non-zero, single-quoted words are not scanned for the history expansion char-
acter. The default value is 0.

2.5 History Programming Example

The following program demonstrates simple use of the GNU History Library.

main ()
{
    char line[1024], *t;
    int len, done = 0;

    line[0] = 0;

    using_history ();
    while (!done)
    {
        printf ("history/$ ");
        fflush (stdout);
        t = fgets (line, sizeof (line) - 1, stdin);
        if (t && *t)
        {
            len = strlen (t);
            if (t[len - 1] == 'n')
                t[len - 1] = '0';
        }

        if (!t)
            strcpy (line, "quit");
if (line[0])
{
    char *expansion;
    int result;

    result = history_expand (line, &expansion);
    if (result)
        fprintf (stderr, "%s\n", expansion);

    if (result < 0 || result == 2)
    {
        free (expansion);
        continue;
    }

    add_history (expansion);
    strncpy (line, expansion, sizeof (line) - 1);
    free (expansion);
}

if (strcmp (line, "quit") == 0)
    done = 1;
else if (strcmp (line, "save") == 0)
    write_history ("history_file");
else if (strcmp (line, "read") == 0)
    read_history ("history_file");
else if (strcmp (line, "list") == 0)
{
    register HIST_ENTRY **the_list;
    register int i;

    the_list = history_list ();
    if (the_list)
        for (i = 0; the_list[i]; i++)
            printf ("%d: %s\n", i + history_base, the_list[i]->line);
}
else if (strncmp (line, "delete", 6) == 0)
{
    int which;
    if ((sscanf (line + 6, "%d", &which)) == 1)
    {
        HIST_ENTRY *entry = remove_history (which);
        if (!entry)
            fprintf (stderr, "No such entry %d\n", which);
        else
        {
            free (entry->line);
            free (entry);
        }
    }
} else {
    fprintf (stderr, "non-numeric arg given to 'delete'\n");
}
Appendix A Concept Index

(Index is nonexistent)
Appendix B  Function and Variable Index

(Index is nonexistent)
# Table of Contents

1 Using History Interactively ................. 1  
   1.1 Interactive History Expansion ............... 1  
   1.1.1 Event Designators .......................... 1  
   1.1.2 Word Designators .......................... 2  
   1.1.3 Modifiers .................................. 2  

2 Programming with GNU History ............... 3  
   2.1 Introduction to History ....................... 3  
   2.2 History Storage .............................. 3  
   2.3 History Functions ............................ 4  
      2.3.1 Initializing History and State Management 4  
      2.3.2 History List Management ................. 4  
      2.3.3 Information About the History List ....... 5  
      2.3.4 Moving Around the History List .......... 5  
      2.3.5 Searching the History List ............... 5  
      2.3.6 Managing the History File ............... 6  
      2.3.7 History Expansion ....................... 6  
   2.4 History Variables ............................ 7  
   2.5 History Programming Example ............... 8  

Appendix A  Concept Index ....................... 11  

Appendix B  Function and Variable Index ....... 13