Introduction

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 making up new ones.

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

The end-user using programs written with the History library has the benifit 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 used by Csh.

If the programmer desires, he can use the Readline library, which includes history manipulation by default, and has the added advantage of Emacs style command line editing.

1 Interactive Use

1.1 History Expansion

The History library provides a history expansion feature that is similar to the history expansion in Csh. The following text describes what syntax features are available.

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*. The line is broken into words in the same fashion that the Bash shell does, so that several English (or Unix) words surrounded by quotes are considered as one word.

1.1.1 Event Designators

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

! Start a history subsititution, except when followed by a `SPC`, `TAB`, `RET`, `=` or `(`.

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

!n Refer to command line *n*.

!-n Refer to the current command line minus *n*.

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

!?string[?] Refer to the most recent command containing *string*.

1.1.2 Word Designators

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 a 0 (zero).

0 (zero) The zero'th word. For many applications, this is the command word.

n The *n*'th 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` is equivalent to `0-y`.

*** All of the words, excepting the zero'th. 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.

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 `:`.

`#` The entire command line typed so far. This means the current command, not the previous command, so it really isn't a word designator, and doesn't belong in this section.

`h` Remove a trailing pathname component, leaving only the head.

`r` Remove a trailing suffix of the form `".xxx"`, leaving the basename (root).

`e` Remove all but the suffix (end).

`t` Remove all leading pathname components (before the last slash), leaving the tail.

`p` Print the new command but do not execute it. This takes effect immediately, so it should be the last specifier on the line.

2 Programming