NAME

history, fc – process command history list

SYNOPSIS

/usr/bin/fc [*first*[*last*]]
/usr/bin/fc –**l** [–**nr**] [*first*[*last*]]
/usr/bin/fc –**s** [*old*=*new*] [*first*]

csh

history [–**hr**] [*n*]

ksh

fc –**e** – [*old*=*new*] [*command*]
fc [–**e** *ename*] [–**nlr**] [*first* [*last*]]

DESCRIPTION

/usr/bin/fc

The **fc** utility lists or edits and reexecutes, commands previously entered to an interactive **sh**.

The command history list references commands by number. The first number in the list is selected arbitrarily. The relationship of a number to its command will not change except when the user logs in and no other process is accessing the list, at which time the system may reset the numbering to start the oldest retained command at another number (usually 1). When the number reaches an implementation-dependent upper limit, which will be no smaller than the value in **HISTSIZE** or 32 767 (whichever is greater), the shell may wrap the numbers, starting the next command with a lower number (usually 1). However, despite this optional wrapping of numbers, **fc** will maintain the time-ordering sequence of the commands. For example, if four commands in sequence are given the numbers 32 766, 32 767, 1 (wrapped), and 2 as they are executed, command 32 767 is considered the command previous to 1, even though its number is higher.

When commands are edited (when the –**l** option is not specified), the resulting lines will be entered at the end of the history list and then reexecuted by **sh**. The **fc** command that caused the editing will not be entered into the history list. If the editor returns a non-zero exit status, this will suppress the entry into the history list and the command reexecution. Any command-line variable assignments or redirection operators used with **fc** will affect both the **fc** command itself as well as the command that results, for example:

fc –s –– –1 2>/dev/null

reinvokes the previous command, suppressing standard error for both **fc** and the previous command.

csh

Display the history list; if *n* is given, display only the *n* most recent events.

–**r** Reverse the order of printout to be most recent first rather than oldest first.

–**h** Display the history list without leading numbers. This is used to produce files suitable for sourcing using the –**h** option to the **csh** built-in command, **source**(1).

History Substitution:

History substitution allows you to use words from previous command lines in the command line you are typing. This simplifies spelling corrections and the repetition of complicated commands or arguments. Command lines are saved in the history list, the size of which is controlled by the **history** variable. The **history** shell variable may be set to the maximum number of command lines that will be saved in the history file; i.e.:

set history = 200

will allow the history list to keep track of the most recent 200 command lines. If not set, the C shell saves only the most recent command.

A history substitution begins with a **!** (although you can change this with the **histchars** variable) and may occur anywhere on the command line; history substitutions do not nest. The **!** can be escaped with **** to suppress its special meaning.

Input lines containing history substitutions are echoed on the terminal after being expanded, but before any other substitutions take place or the command gets executed.

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 character, tab, newline, **=** or **(**.

!! Refer to the previous command. By itself, this substitution repeats the previous command.

!*n* Refer to command line *n*.

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

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

!?*str***?** Refer to the most recent command containing *str*.

!?*str***?** *additional*
Refer to the most recent command containing *str* and append *additional* to that referenced command.

!{*command***}** *additional*
Refer to the most recent command beginning with *command* and append *additional* to that referenced command.

*^previous_word^replacement^*
Repeat the previous command line replacing the string *previous_word* with the string *replacement*. This is equivalent to the history substitution:
!:s/*previous_word***/***replacement***/**.

To re-execute a specific previous command AND make such a substitution, say, re-executing command #6,
!:6s/*previous_word***/***replacement***/**.

Word Designators:

A '**:**' (colon) separates the event specification from the word designator. It can be omitted if the word designator begins with a **^**, **$**, *****, **–** or **%**. If the word is to be selected from the previous command, the second **!** character can be omitted from the event specification. For instance, **!!:1** and **!:1** both refer to the first word of the previous command, while **!!$** and **!$** both refer to the last word in the previous command. Word designators include:

The entire command line typed so far.

0 The first input word (command).

n The *n*'th argument.

- **^** The first argument, that is, **1**.
- **$** The last argument.
- **%** The word matched by (the most recent) **?***s* search.
- *x*–*y* A range of words; –*y* abbreviates **0**–*y*.
- ***** All the arguments, or a null value if there is just one word in the event.
- *x** Abbreviates *x*–**$**.
- *x*– Like *x** but omitting word **$**.

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 the head.
- **r** Remove a trailing suffix of the form '**.***xxx*', leaving the basename.
- **e** Remove all but the suffix, leaving the Extension.
- **s/***oldchars***/***replacements***/**
  Substitute *replacements* for *oldchars*. *oldchars* is a string that may contain embedded blank spaces, whereas *previous_word* in the event designator
  ^*oldchars*^*replacements*^
  may not.
- **t** Remove all leading pathname components, leaving the tail.
- **&** Repeat the previous substitution.
- **g** Apply the change to the first occurrence of a match in each word, by prefixing the above (for example, **g&**).
- **p** Print the new command but do not execute it.
- **q** Quote the substituted words, escaping further substitutions.
- **x** Like **q**, but break into words at each space character, tab or newline.

Unless preceded by a **g**, the modification is applied only to the first string that matches *oldchars*; an error results if no string matches.

The left-hand side of substitutions are not regular expressions, but character strings. Any character can be used as the delimiter in place of **/**. A backslash quotes the delimiter character. The character **&**, in the right hand side, is replaced by the text from the left-hand-side. The **&** can be quoted with a backslash. A null *oldchars* uses the previous string either from a *oldchars* or from a contextual scan string *s* from **!?***s*. You can omit the rightmost delimiter if a newline immediately follows *replacements*; the rightmost **?** in a context scan can similarly be omitted.

Without an event specification, a history reference refers either to the previous command, or to a previous history reference on the command line (if any).

ksh Using **fc**, in the form of

fc –**e** – [*old***=***new*] [*command*],

the *command* is re-executed after the substitution *old***=***new* is performed. If there is not a *command* argument, the most recent command typed at this terminal is executed.

Using **fc** in the form of

fc [–**e** *ename*] [–**nlr**] [*first* [*last*]],

a range of commands from *first* to *last* is selected from the last **HISTSIZE** commands that were typed at the terminal. The arguments *first* and *last* may be specified as a number or as a string. A string is used to locate the most recent command starting with the given string. A negative number is used as an offset to the current command number. If the –**l** flag is selected, the commands are listed on standard output. Otherwise, the editor program –**e** *name* is invoked on a file containing these keyboard commands. If *ename* is not supplied, then the value of the variable **FCEDIT** (default **/bin/ed**) is used as the editor. When editing is complete, the edited command(s) is executed. If *last* is not specified then it will be set to *first*. If *first* is not specified the default is the previous command for editing and –16 for listing. The flag –**r** reverses the order of the commands and the flag –**n** suppresses command numbers when listing. (See **ksh**(1) for more about command line editing.)

HISTFILE

If this variable is set when the shell is invoked, then the value is the pathname of the file that will be used to store the command history.

HISTSIZE

If this variable is set when the shell is invoked, then the number of previously entered commands that are accessible by this shell will be greater than or equal to this number. The default is **128**.

Command Re-entry:

The text of the last **HISTSIZE** (default 128) commands entered from a terminal device is saved in a **history** file. The file **$HOME/.sh_history** is used if the **HISTFILE** variable is not set or if the file it names is not writable. A shell can access the commands of all *interactive* shells which use the same named **HISTFILE**. The special command **fc** is used to list or edit a portion of this file. The portion of the file to be edited or listed can be selected by number or by giving the first character or characters of the command. A single command or range of commands can be specified. If you do not specify an editor program as an argument to **fc** then the value of the variable **FCEDIT** is used. If **FCEDIT** is not defined then **/bin/ed** is used. The edited command(s) is printed and re-executed upon leaving the editor. The editor name – is used to skip the editing phase and to re-execute the command. In this case a substitution parameter of the form *old*=*new* can be used to modify the command before execution. For example, if **r** is aliased to **’fc –e –’** then typing **‘r bad=good c’** will re-execute the most recent command which starts with the letter **c**, replacing the first occurrence of the string **bad** with the string **good**.

Using the **fc** built-in command within a compound command will cause the whole command to disappear from the history file.

OPTIONS

The following options are supported:

–**e** *editor* Use the editor named by *editor* to edit the commands. The *editor* string is a utility name, subject to search via the **PATH** variable. The value in the **FCEDIT** variable is used as a default when –**e** is not specified. If **FCEDIT** is null or unset, **ed** will be used as the editor.

–l (The letter ell.) List the commands rather than invoking an editor on them. The commands will be written in the sequence indicated by the *first* and *last* operands, as affected by **–r**, with each command preceded by the command number.

–n Suppress command numbers when listing with **–l**.

–r Reverse the order of the commands listed (with **–l)** or edited (with neither **–l** nor **–s**).

–s Re-execute the command without invoking an editor.

OPERANDS

The following operands are supported:

first
last Select the commands to list or edit. The number of previous commands that can be accessed is determined by the value of the **HISTSIZE** variable. The value of *first* or *last* or both will be one of the following:

[+]*number* A positive number representing a command number; command numbers can be displayed with the **–l** option.

–number A negative decimal number representing the command that was executed *number* of commands previously. For example, –1 is the immediately previous command.

string A string indicating the most recently entered command that begins with that string. If the *old=new* operand is not also specified with **–s**, the string form of the *first* operand cannot contain an embedded equal sign.

When the synopsis form with **–s** is used:

- If *first* is omitted, the previous command will be used.

For the synopsis forms without **–s :**

- If *last* is omitted, *last* defaults to the previous command when **–l** is specified; otherwise, it defaults to *first*.
- If *first* and *last* are both omitted, the previous 16 commands will be listed or the previous single command will be edited (based on the **–l** option).
- If *first* and *last* are both present, all of the commands from *first* to *last* will be edited (without **–l)** or listed (with **–l**). Editing multiple commands will be accomplished by presenting to the editor all of the commands at one time, each command starting on a new line. If *first* represents a newer command than *last*, the commands will be listed or edited in reverse sequence, equivalent to using **–r .** For example, the following commands on the first line are equivalent to the corresponding commands on the second:

  fc –r 10 20 **fc 30 40**

  fc 20 10 **fc –r 40 30**

- When a range of commands is used, it will not be an error to specify *first* or *last* values that are not in the history list; **fc** will substitute the value representing the oldest or newest command in the list, as appropriate. For example, if there are only ten commands in the history list, numbered 1 to 10:

fc –l

fc 1 99

will list and edit, respectively, all ten commands.

old=new Replace the first occurrence of string *old* in the commands to be reexecuted by the string *new*.

OUTPUT

When the **–l** option is used to list commands, the format of each command in the list is as follows:

"%d\t%s\n", <*line number***>, <***command***>**

If both the **–l** and **–n** options are specified, the format of each command is:

"\t%s\n", <*command***>**

If the *command* consists of more than one line, the lines after the first are displayed as:

"\t%s\n", <*continued-command***>**

EXAMPLES

csh	ksh
% history	$ fc -l
1 cd /etc	1 cd /etc
2 vi passwd	2 vi passwd
3 date	3 date
4 cd	4 cd
5 du .	5 du .
6 ls -t	6 ls -t
7 history	7 fc -l
% !d	$ fc -e - d
du .	du .
262 ./SCCS	262 ./SCCS
336 .	336 .
% !da	$ fc -e - da
Thu Jul 21 17:29:56 PDT 1994	Thu Jul 21 17:29:56 PDT 1994
%	$ alias \!='fc -e -'
% !!	$!
date	alias ='fc -e -'
Thu Jul 21 17:29:56 PDT 1994	
% !6	$! 6

du . **262 ./SCCS** **336 .**	**du .** **262 ./SCCS** **336 .**
% !ls ma∗ **ls -t malloc.c** **malloc.o** **malloc.c**	**$! ls ma∗** **ksh: !l: not found**

ENVIRONMENT

See **environ**(5) for descriptions of the following environment variables that affect the execution of **fc**: **LC_CTYPE**, **LC_MESSAGES**, and **NLSPATH**.

FCEDIT This variable, when expanded by the shell, determines the default value for the **e** *editor* option's *editor* option-argument. If **FCEDIT** is null or unset, **ed** will be used as the editor.

HISTFILE Determine a pathname naming a command history file. If the **HISTFILE** variable is not set, the shell may attempt to access or create a file **.sh_history** in the user's home directory. If the shell cannot obtain both read and write access to, or create, the history file, it will use an unspecified mechanism that allows the history to operate properly. (References to history ''file'' in this section are understood to mean this unspecified mechanism in such cases.) An implementation may choose to access this variable only when initialising the history file; this initialisation will occur when **fc** or **sh** first attempt to retrieve entries from, or add entries to, the file, as the result of commands issued by the user, the file named by the **ENV** variable, or implementation-dependent system startup files. (The initialisation process for the history file can be dependent on the system startup files, in that they may contain commands that will effectively preempt the user's settings of **HISTFILE** and **HISTSIZE**. For example, function definition commands are recorded in the history file, unless the **set** −**o nolog** option is set. If the system administrator includes function definitions in some system startup file called before the **ENV** file, the history file will be initialised before the user gets a chance to influence its characteristics.) In some historical shells, the history file is initialised just after the **ENV** file has been processed. Therefore, it is implementation-dependent whether changes made to **HISTFILE** after the history file has been initialised are effective. Implementations may choose to disable the history list mechanism for users with appropriate privileges who do not set **HISTFILE**; the specific circumstances under which this will occur are implementation-dependent. If more than one instance of the shell is using the same history file, it is unspecified how updates to the history file from those shells interact. As entries are deleted from the history file, they will be deleted oldest first. It is unspecified when history file entries are physically removed from the history file.

HISTSIZE Determine a decimal number representing the limit to the number of previous commands that are accessible. If this variable is unset, an unspecified default greater than or equal to 128 will be used. The maximum number of commands in the history list is unspecified, but will be at least 128. An implementation may choose to access this variable only when initialising the history file, as described under **HISTFILE**. Therefore, it is unspecified whether changes made to **HISTSIZE** after the history file has been initialised are effective.

EXIT STATUS

The following exit values are returned:

0 Successful completion of the listing.

>0 An error occurred.

Otherwise, the exit status will be that of the commands executed by **fc**.

SEE ALSO

csh(1), **ksh**(1), **sh**(1), **source**(1), **environ**(5)