# Contents

**Product Specification** . . . . . 1
**Control Structures** . . . . . 2
Basic Statements 2
Condition Testing 2
Iteration 3
Procedures 3
Aliases 4
Variable and Expression Manipulation 4
Local Variables 5
Function Keys 6
Miscellaneous Statements 7
**Built-in Procedures** . . . . . 8
**Utilities** . . . . . 10
**Shell Features** . . . . . 15
History Recall 15
Command Completion 15
Wildcarding and Pattern Matching 16
Filename Completion 16
Command Line Editing 17
**Expressions** . . . . . 18
Statement Relationships 18
I/O Redirection 18
Quoting 19
Escape Sequences 19
Expression Operators 20
File System Tests 21
Variable Substitution 22
Substitution Modifiers 23
Pathname Editing 23
**Predefined Variables** . . . . . 24
**Starting Hamilton C shell** . . . . . 28
**Regular Expressions** . . . . . 29
**Sample Applications** . . . . . 30

# Hamilton C shell™ Quick Reference

## Product Specification:

Provide full compliance with the entire C shell language (except job control) as defined in the Berkeley 4.3 *Unix® Programmer's Manual* and by Anderson & Anderson in *The UNIX C Shell Field Guide.*

Provide a complete set of all the important utilities popular on high-end workstations including such favorites as **grep, sed, head, tail, diff, ls, more, mv, cp, rm** and many others.

Design everything from scratch for OS/2® protected mode:

1. Show off all the best of OS/2: HPFS, long filenames and extended attributes; networks; text, full-screen and PM applications; highlighting and color; standard OS/2 conventions for key bindings and environmental variables.

2. Provide world-class features: history and command line editing of enormous command lines with arrow and function keys; filename and command completion; wildcarding; piping and command substitution; background activities; aliases, procedures and local variables.

3. Use a modern top-down parser for better language recognition. Allow control structures such as **foreach** or **if** to be nested arbitrarily, piped or put in the background.

4. Take advantage of OS/2 threads to achieve performance and functionality not possible in UNIX.

5. Make it responsive: very fast interrupts, command line editing, screen updates and spawning of children.

Provide the highest possible performance, especially when executing shell scripts or iterative statements.

Provide fanatical quality.

**Hamilton Laboratories,** 13 Old Farm Road, Wayland, MA 01778, 508-358-5715

# Hamilton C shell Quick Reference

## Control Structures

### Basic Statements:

Same as `cmd.exe`: a file reference + arguments.

Examples:
```
cl -AS -G2 -Zi hello.c
cp hello.exe c:\os2\bin
```

Hamilton C shell maintains a hash structure which allows it to quickly search for a suitable `.csh`, `.exe`, `.com` or `.cmd` file (in that order) in each of as many as 256 path directories. Wildcarding is done by the shell before invoking the child. Up to 64K of environmental and 64K of command-line argument data (the limits of OS/2) can be passed to a child process.

### Condition-Testing:

**if** ( <expr> ) **then**
    <statement_list>
**else**
    <statement_list>
**end**

**if** ( <expr> ) <statement>

**switch** ( <expr> )
    **case** <expr> :
        <statement_list>
    **case** <expr> :
        <statement_list>
    **default** :
        <statement_list>
**end**

Where an expression is expected, a conventional high level language syntax is accepted: e.g., names refer to variables, '*' means multiply, not wildcard and '>' means greater than, not i/o redirection.

The short form of the **if** statement dispenses with the alternate **else** case. Type the whole thing on one line. In a **switch** statement, expressions are compared by pattern match: the **case** expression can be a string with wildcard characters. Comparisons are made down the list of alternatives until one matches. All following statements are executed until a **break** is encountered.

## Iteration:

**foreach** <name> ( <word list> )
    <statement_list>
**end**

**for** <name> = <expr> [ **to** <expr> ] [ **by** <expr> ] **do**
    <statement_list>
**end**

**while** ( <expr> )
    <statement_list>
**end**

**repeat** <number> <statement>

**repeat**
    <statement_list>
**until** ( <expr> )

The **foreach** statement is intended for iteration over a list of words, often specified by wildcarding. The **for** statement offers the more conventional numeric iteration. Multiple iteration ranges, separated by commas, can be specified on the **for** statement.

## Procedures:

**proc** <name> ( [ <namelist> ] )
    <statement_list>
    **return** [ <expr> ]
**end**

**proc**

**unproc** <namelist>

Procedures defined by the **proc** statement can recursively call other procedures. They can be referred to inside an expression or as a new command, in which case any value returned is written to stdout. The **proc** statement with no arguments causes a list of the available procedures to be written. The **unproc** statement allows a procedure to be discarded.

## Aliases:

**alias** <name> [ = ] ( <word list> )
**alias** <name> [ = ] <word list>

**alias**
**alias** <name>

**unalias** <namelist>

Aliases can be referred to at the beginning of a command and provide a quick, user-defined shorthand. **alias** <name> with no arguments prints the value of the name. **alias** without any arguments prints the values of all aliases.

## Variable and Expression Manipulation:

@ <expr>
**calc** <expr>

The @ and **calc** statements will each calculate the value of an expression; the @ statement does it silently while the **calc** statement writes the result to stdout.

**set** <named_ref> [ = ] ( <word list> )
**set** <named_ref> [ = ] <word list>
**setenv** <named_ref> [ = ] ( <word list> )
**setenv** <named_ref> [ = ] <word list>
**shift** [ <name> ]

**set**
**set** <name>
**setenv**
**setenv** <name>

**unset** <namelist>
**unsetenv** <namelist>

The **set**, **setenv** and **shift** statements manipulate variables as words rather than expressions. **set** defines a set variable that's shared between all threads in the shell; **setenv** puts it into the environment and inherited by child processes. **set** or **setenv** with no operands prints a list of all defined variables of that type. **set** <name> or **setenv** <name> with no arguments print the value of the named variable. **unset** or **unsetenv** let you discard a variable.

## Local Variables:

The **local** command lets you define a list of variable names that you don't to share with other routines or other threads or processes. When you define a local variable it hides any previous definition from any outer statement list. (But you are not permitted to redefine any of the built-in **set** or **setenv** variable names.)

**local** <namelist>
**local**

The <namelist> should be typed with commas between the names. When you create a new local variable, its initial value is always a null string. Typing **local** with no operands reports the currently defined and accessible local variables, if any.

Local variables are automatically discarded as soon as execution leaves the statement nesting level in which the variable was created. You can also explicitly discard local variables using the **unlocal** command.

**unlocal** <namelist>

In all other respects, local variables act just like any other variables, though you may find they're slightly faster since the shell doesn't need to semaphore its use of them.

# Function Keys

## setkey command:

The **setkey** command lets you define a list of words that should be stuffed back onto the command-line whenever you press a particular function key. The syntax is exactly the same as used in the **set**, **setenv** and **alias** commands:

**setkey** <fkey> [ = ] ( <word list> )
**setkey** <fkey> [ = ] <word list>

where <fkey> is any of the function keys **f1** (or **F1**) through **f12** (or **F12**.)

Typing **setkey** with no operands reports the current function key bindings, if any. Also, a corresponding **unsetkey** command lets you discard key bindings:

**setkey**
**unsetkey** <fkeylist>

The <fkeylist> should be typed with commas between the keys. For example:

```
unsetkey f1, f2
```

## Using the Function Keys

| Key | Meaning |
|---|---|
| <F*x*> | Clear the command line, post the text bound to this key and execute the command. |
| Alt-<F*x*> | Insert the text bound to this key at the cursor location but don't execute it yet. |
| Ctrl-<F*x*> | Clear the command line and post the text bound to this key but don't execute it yet. |

Since the function key's bound text is written back into the command line inside command line editor, the substitution happens ahead of any parsing of the command line into words or expansion of history "!..." or "%..." references so it is possible to meaningfully embed these kinds of references into the key binding.

## Miscellaneous Statements

| Statement | Function |
|---|---|
| <drive>: | Change current drive. |
| <label>: <statement> | Define a label. |
| ( <statement_list> ) | Group a list of statements, saving and restoring the current directory during execution |
| **break** [ <name> ] | Exit from the named or, by default, the innermost **switch, foreach, for, while** or **repeat** statement. |
| **continue** [ <name> ] | Continue with the next iteration of the named or innermost **foreach, for, while** or **repeat**. |
| **exit** [ <expr> ] | Exit from this thread or, if this is the main thread, from the C shell. |
| **goto** <name> | Continue at the labeled statement. |
| **onintr** <statement> | Define the action to be taken if an interrupt is signaled. |
| **source** <wordargs> | Read and process statements from a file as if they were typed into this thread. |
| **time** <statement> | Execute the statement and report how long it took. |
| # | Comment text up to the end of the line. |

# Built-in Procedures

| Name | Function |
|---|---|
| **Filename Functions:** | |
| childpath(*p*, *c*) | Test whether filename *c* could be in a subdirectory of *p*. (Does not test for actual existence of either *c* or *p*.) |
| driveno(*p*) | Drive number implied by pathname *p*. |
| fullpath(*p*) | Fully resolve pathname *p*. |
| samepath(*a*, *b*) | Test whether two filenames, *a* and *b*, point to the same file. |
| **Math Functions:** | |
| abs(*x*) | Absolute value |
| acos(*x*) asin(*x*) atan(*x*) cos(*x*) sin(*x*) tan(*x*) | Trigonometric functions |
| cosh(*x*) sinh(*x*) tanh(*x*) | Hyperbolic functions |
| ceil(*x*) | Ceiling (lowest integer ≥ *x*) |
| exp(*x*) log(*x*) log2(*x*) log10(*x*) | Exponential and logarithmic functions |
| floor(*x*) | Floor (highest integer ≤ *x*) |
| round(*x*) | floor(*x* + 0.5) |
| sqrt(*x*) | Square root |
| **String Functions:** | |
| char(*i*) | Return the character corresponding to the numeric value *i*. |
| code(*c*) | Return the numeric encoding of the character *c*. |
| concat(*a*, *b*, ...) | Concatenation of a series of strings. |
| isinteger(*x*) | Test whether *x* is an integer. (Remember that null strings and strings consisting only of white space are considered equal to 0.) |
| isnumber(*x*) | Test whether *x* is a number. |

## Built-in Procedures

| Name | Function |
|---|---|
| printf(*fmt*, ...) | Perform C language-style print formatting, returning the result as a string. These argument formats are recognized:<br>%c Single character.<br>%d Decimal number.<br>%e [-]d.dddddde[+-]ddd<br>%f [-]ddd.dddddd<br>%g %e or %f formatting, whichever is shorter.<br>%o Unsigned octal number.<br>%s String.<br>%x Unsigned hexadecimal number.<br>%% Literal % character.<br>Additional parameters may lie between the % and the control letter:<br>- Left-justify expression in its field.<br>width Pad field to this width as needed; leading 0 pads with zeros.<br>.prec Maximum string width or digits to right of decimal point. |
| reverse(*s*) | Reverse the order of characters in *s*. |
| strindex(*a*, *b*) | Return the position in *a* of the first occurrence of *b*. (0 means *b* was not found.) |
| strlen(*s*) | Number of characters in *s*, represented as a string. |
| substr(*s*, *b*, *i*) | Substring of length *i* beginning at *b*-th character of *s*. (*b* = 1 is the first character; *i* = 0 means "rest of *s*.") |
| upper(*s*) lower(*s*) | Translate a string to all upper- or all lower-case. |

## Utilities

In this table, *italics* indicates a built-in utility. Normal typestyle indicates an external utility. `Courier` indicates an alias.

All utilities self-document with the `-h` option. Any external utility may be renamed simply by renaming the executable file. Additional utilities are planned and will be sent free to registered users.

| Command | Function |
|---|---|
| `app` | Append to a file. |
| ar2 | Archive or restore OS/2 files and directories. Similar to tar but designed for long filenames, extended attributes and other OS/2-specific characteristics. |
| `beep` | Beep sound. |
| cat | Concatenate files. |
| *cd* | Change current directory. Optionally, change disk. |
| `cdd` | Change both current directory and current disk. |
| *chcp* | Change code page. |
| *chdir* | A synonym for cd. |
| chmod | Change mode bits (Hidden, System, Read-Only and Archive) of file. Optionally, recursively walk through directories, chmod'ing all the contents. |
| *cls* | Clear the screen. |
| `copy` | Invoke the standard IBM/MS copy command with shell wildcarding turned off so copy will work sensibly. |
| cp | Copy files or directories. Options for interactive and logging modes and for merging sub-directories. |
| csh | Invoke Hamilton C shell. |
| cut | Cut out selected fields of each line of text. Fields can be defined by delimiter characters or by column numbers. |
| `date` | Display the current time and date using dt.exe. |
| `del` | Delete files. |
| `dir` | Invoke the cmd.exe dir command. |
| diff | Compare files or directories. Optionally generates merged listings of old and new versions, showing changes in context using color highlighting. Options for ignoring differences in white space or character case and for setting the re-synchronization window size. |

## Utilities

| Command | Function |
|---|---|
| dim | Discard any ansi escape sequences in the input stream. |
| *dirs* | Print the directory stack. |
| `dskdup` | Fast duplication of a diskette using the dskread and dskwrite utilities. |
| dskread | Read low-level sectors from a disk to stdout. |
| dskwrite | Write low-level sectors from stdin to a disk. |
| dt | Display the date and time. |
| du | Display disk usage. Shows amount and percentages of allocated and free space in a partition. Optionally shows cluster information. |
| `dumphist` | Dump out the history list. |
| *echo* | Echo arguments to stdout. Options for writing to stderr instead of stdout and for omitting any trailing newline. |
| `erase` | Older IBM/Microsoft name for deleting a file. |
| *eval* | Reparse and execute the argument word list as a command after any run-time substitutions or wildcarding. |
| `f` | Quicker name for fgrep. |
| fgrep | Fast string search (fast grep) of text files. Can search for any number of strings in a single pass. Options for ignoring differences in white space or character case, reporting line numbers, etc. |
| find | Find all files in a directory matching certain criteria. |
| `g` | Quicker name for grep. |
| grep | Regular expression pattern search of text files. Includes fgrep-style option for searching for any number of patterns in a single pass. Options for ignoring character case, reporting line numbers, etc. |
| `h` | Quicker name for history. |
| *hashstat* | Print path hash statistics. Tells how many tries, on the average, the shell needs to find a file in the PATH directories. (Usually less than 2 tries.) |
| head | Copy the first few lines or bytes of a file to stdout. Optional tab expansion. |

## Utilities

| Command | Function |
|---|---|
| help | Invoke the IBM/Microsoft help command. |
| *history* | Display the history list of past commands. |
| `home` | Change to the home disk and directory. |
| *kill* | Kill background threads, processes or screens. |
| label | Read/Write the volume label. |
| `ll` | List directories, long format. |
| `loadhist` | Load the history list without executing any of it. |
| ls | List directory contents. Options for selecting only certain types of files, recursively walking through entire directory trees listing contents or summing file sizes, and sorting and displaying the results in a number of formats. |
| markexe | Mark an `.exe` file to indicate whether an application is text-windowable, full-screen or PM graphics and whether it supports long filenames. |
| `md` | Make directories. |
| `mi` | Quick interactive startup of your favorite version more. Clears the screen when it starts up and doesn't just exit if there's less than a screenful. |
| `mih` | Huge interactive more. |
| `mis` | Small interactive more. |
| mkdir | Make a new directory. |
| more | A better more utility. Able to search forward or backward or to a specific line and to display non-printable characters in binary or as C language-style escapes. On-line help. |
| moreh | A large model version of more. Not quite as fast, but able to remember megabytes of data coming through a pipe. |
| mv | Move files or directories. Options for interactive and logging modes and for merging sub-directories. |
| newer | Test whether first file is newer than the others. |
| older | Test whether first file is older than the others. |
| patchlnk | A (very) special-purpose utility to patch a bug in the Microsoft linker. |
| `pause` | Pause, waiting for any keystroke or character from stdin. |

## Utilities

| Command | Function |
|---|---|
| *popd* | Pop directory stack. |
| *ps* | List process and thread status. |
| *pushd* | Push a new current directory on the directory stack or exchange the top two items. |
| pwd | Print the working directories. |
| `q` | Exit the C shell. |
| `rd` | Remove empty directories. |
| *rehash* | Rehash the path directories. |
| `ren` | Another name for the rename alias. |
| `rename` | Invoke the standard IBM/MS rename command with shell wildcarding turned off so the rename will work sensibly. |
| rm | Remove files or directories. Options for removing entire directory trees or even read-only or system files and directories. |
| rmdir | Remove empty directories. |
| *rotd* | Rotate the directory stack. |
| sed | Stream editor. Reads a line at a time from stdin, does whatever editing is requested and writes the result to stdout. Provides search and replace with regular expressions, character translations, inserting and deleting blocks of text and branching and condition-testing. |
| setrows | Set or report the number of rows in the display window. |
| *sleep* | Sleep for a specified period. |
| sort | A better, faster sort, capable of handling more than 64K bytes. Options for sorting on just certain fields, etc. |
| *source* | Read commands from a file. |
| split | Split a large file into equal-sized chunks counting either by bytes or lines. |
| `start` | Start a new session. |
| strings | Extract ASCII strings from a binary file. Options for setting minimum string lengths, whether they need a line end or a null character at the end, and reporting offsets where the strings were found. |
| sum | Checksum the contents of a file. Options for several checksum algorithms. |

## Utilities

| Command | Function |
|---|---|
| tabs | Expand/Unexpand tabs. |
| tail | Copy the last few lines or bytes of a file to stdout. Includes -f (follow) option for watching as data is added to the end of a file by another process. Optional tab expansion. |
| tar | Read/Write UNIX tape archive (tar) format files. Options for interatively or automatically renaming files as necessary for HPFS or FAT partitions, swapping byte sex (including auto-sensing byte sex in an archive), selecting just a portion or an archive and converting between UNIX and OS/2 line end conventions. |
| tee | Pipe fitting. Snapshot data passing through a pipe into one or more files. |
| touch | Update the time-stamp on a file or a directory. Recursive option for touching everything in a directory. |
| tr | Translate characters. Options for editing out specified characters or just repeated characters and for normalizing line endings. |
| `type` | Copy files to stdout. |
| *unhash* | Turn off path hashing. |
| uniq | Unique lines: discard adjacent duplicates. Options for ignoring white space and for reporting only lines with duplicates or only lines with no duplicates. |
| ver | Display the current OS/2 and Hamilton C shell version numbers. |
| *verify* | Turn write verification mode on or off. Write verification on means the OS/2 kernel will be asked to always verify that any data written to a disk can be read. |
| `vol` | List volume labels using vl.exe. |
| *wait* | Wait for children to complete. |
| wc | Count lines, words and characters. |
| whereis | Tell which PATH directory a given executable is in. |
| `xcopy` | Invoke the standard IBM/MS xcopy command with shell wildcarding turned off so xcopy will work sensibly. |
| xd | Hex dump a file to stdout. Options for specifying offsets to start and stop dumping, binary and floating point formats, arbitrary radix. |

## History Recall

History recall allows a previous statement to be quickly recalled and re-executed. It's a very fast shorthand, especially in the edit/compile/debug loop or to fix a typo. For convenience, "!" is taken as an ordinary character if followed by white space, "=", "~" or "(".

If you want, you can choose different characters to introduce history references by changing the `histchars` variable.

| Command | Meaning |
|---|---|
| !! | Last command |
| !^ | First argument word of last command |
| !$ | Last word of last command |
| !* | All arguments of last command |
| !**n** | Command **n** |
| !-**n** | **n**th command from the last |
| !**str** | Last command starting with **str** |
| !?**str**? | Last command containing **str** |
| %**str1**%**str2**% | Substitute **str2** for **str1** in last command. (Used only at the beginning of a line.) |

## Command Completion

Command completion lets you type just part of a previous command and have the shell fill in the rest. As with filename completion, bright red highlighting is used if no match is found. Consecutive depressions cause the search to continue on back through the history list.

| Key | Meaning |
|---|---|
| Ctrl-<Enter> | Search for the last command that starts with the characters in the previous word. |
| Alt-<Enter> | Search for the last command that contains the characters in the previous word anywhere on the command line. |

# Hamilton C shell Quick Reference

## Wildcarding and Pattern Matching

Wildcarding is nestable arbitrarily and uses a recursive comparison algorithm to guarantee a sensible result no matter how complex the pattern. For example: `*r*` or even `*\[a-c]*.[ch]` operate sensibly. Wildcards will match any filename except "." and ".." unless it's marked "hidden."

| Characters | Meaning |
|---|---|
| ? | Match any single character, including '.' but not '\' or '/'. |
| * | Match any number of characters, including '.' but not '\' or '/'. |
| [a-z] | An example range: match any character **a** through **z**. |
| [^a-z] | An example exclusion range: match any character **not** in the set **a** through **z**. |
| {a,b}c | Alternation: generate both **ac** and **bc**. |

## Filename Completion

Filename completion lets you type just the first part of a filename and have the shell fill in the rest. The two variations are using the F key for basic filename completion or the D key if you want all the duplicates listed.

| Key | Meaning |
|---|---|
| Alt-F or Cntl-F | **Filename completion.** Appending the "*" wildcard character onto the end, use the previous word as a wildcard pattern. If it matches a single file, substitute it in with a space following.<br><br>If there were multiple matches, but they all had some common front-part that fully "used up" the pattern, substitute in just that common front-part and show it in green.<br><br>If substitution wasn't possible, highlight the pattern in bright red. (Any highlighting color is turned when you press the next keystroke.) |
| Alt-D or Cntl-D | **Duplicate completions.** Same wildcarding, but if there are multiple matches, show them all with a space following. If there were no matches, highlight the pattern with bright red. |

# Hamilton C shell Quick Reference

## Command Line Editing

| Key | Meaning |
|---|---|
| <Enter> | Accept the command as typed. Move to the end (if not there already) and carriage return to a new line. |
| <Home> | Beginning of command line. |
| <End> | End of command line. |
| ↑ | Up one command in the history list. Each time it's pressed, it displays the preceding entry in the history list. Any "!..." or "%..." history references in the original text will have been fixed up unless it was the immediately preceding command and it had one these references that failed. If already at the first entry, the command line is highlighted in bright red. |
| ↓ | Down one command line in the history list. If already at the latest entry, the command line is highlighted in bright red. |
| ← | One character left. |
| → | One character right. |
| Ctrl-<Home> | Move to the upper-leftmost character in the current screenful if the command is long enough that it actually wraps across several screens. |
| Ctrl-<End> | Move to the lower-rightmost character in the current screenful. |
| Ctrl-↑ | Up one row on the screen if the command is long enough that it runs over a row. |
| Ctrl-↓ | Down one row on the screen. |
| Ctrl-← | Backup word. |
| Ctrl-→ | Forward word. |
| Alt-<Home> | Delete all preceding characters on the command line. |
| Alt-<End> | Delete all following characters. |
| Alt-↑ | Delete up one row on the screen if the command runs over a row. |
| Alt-↓ | Delete down one row. |
| Alt-←<br>Ctrl-<Backspace> | Delete preceding word. |
| Alt-→ | Delete following word. |
| <Insert> | Toggle insert/overstrike mode. When inserting, the cursor is slightly thicker. |
| Ctrl-<Insert> | Insert the next word from the last section of deleted text. When it reaches the end of the deleted text, it starts over. |
| Alt-<Insert> | Insert all the rest of the previously deleted text. |
| <PageUp> | Backup to one past the last history reference. (Repeatedly typing <PageUp> <Enter> is a convenient way of picking up a whole series of commands from history.) |
| <PageDown> | Forward to the newest entry in the history list. |
| <Esc> | Clear the command line. |

***Note: Users lacking separate arrow keys must press Ctrl-Shift instead of Alt.***

## Statement Relationships

The grammar is completely recursive, so statements of arbitrary complexity can be freely nested, conditionally executed, piped or redirected.

In order of decreasing precedence:

| Operator | Meaning |
|---|---|
| ( ) | Grouping |
| > >! >& >&! >> >>! >>& >>&! < << | I/O Redirection |
| \| \|& | Piping (stdout only or stdout + stderr) between concurrent operations |
| ... & | Background thread or process |
| && \|\| | Conditional execution: only if first succeeds or only if first fails |
| ; | Serial execution |

## I/O Redirection

| Operator | Meaning |
|---|---|
| > >! >& >&! | Output to a file. '!' allows an existing file to be overwritten even if noclobber is set. '&' redirects both stdout and stderr. |
| >> >>! >>& >>&! | Append to a file |
| < | In from a file |
| << <string> | Inline data: the text on the following lines, up to the line containing only the specified <string> will be fed as stdin to the statement. |

## Quoting

| String | Meaning |
|---|---|
| '...' | **Literal character string.** Only do history substitutions. |
| "..." | **Single word.** Typically used if there are embedded blanks or wildcard characters you want treated as ordinary. Has no effect on command or variable substitutions: they're still done. |
| `...` | **Command substitution.** Evaluate the string as a separate command and substitute its output back onto the command line. Newlines are turned into spaces and Ansi escape sequences (for highlighting, etc.) are filtered out. |
| ^ | **Quote just the next character.** Use to remove any special meaning from the next character, to specify a character by its binary value or to specify one following non-printable characters. If the NewLine character at the end of a line is quoted this way, it's treated as ordinary white space. (You can choose a different escape character by changing the escapesym variable.) |

## Escape Sequences

| String | Meaning |
|---|---|
| ^a | Audible alert (bell) |
| ^b | Backspace |
| ^f | Form Feed |
| ^n | New Line |
| ^r | Carriage Return |
| ^t | Tab |
| ^v | Vertical Tab |
| ^^ | Single escapesym character |

## Expression Operators

In order of decreasing precedence:

| Operator | Meaning |
|---|---|
| ( ) | Grouping or Procedure call arguments |
| { } | Run the enclosed statement list and return 1 if it succeeds or 0 otherwise. |
| [ ] | Array indexing. (The first element is element 0.) |
| -A -D -H -R -S -d -e -f -o -w -x -z | File system tests |
| ++ -- | Prefix and postfix increment/decrement |
| ~ - ! + | Bitwise, arithmetic and logical complements and unary plus |
| ** | Exponentiation |
| * / % | Multiplication, Division and Remainder |
| + - | Addition and Subtraction |
| << >> | Bit Shifting |
| == != =~ !~ < <= >= > | Relation-testing and pattern-matching operators |
| & | Bit And |
| ^ | Bit Xor |
| \| | Bit Or |
| && | Logical And |
| \|\| | Logical Or |
| ?: | Conditional selection |
| = += -= *= /= %= >>= <<= &= ^= \|= **= | Assignment operators |

Expressions result in sensible types, considering both the types *and* the values of the operands. For example, 10/2 returns the integer 5 but 5/2 produces the floating point value 2.5. Also, the integer 1, the floating point value 1.0 and the string "1" all compare equal.

## File System Tests

The operand of a file system test is interpreted as a word, not an expression, and may involve wildcarding. If wildcarding produces more than one match, the test is done on the first one.

| Prefix Operator | True if |
|---|---|
| -A | Archive Bit Set |
| -D -d | Directory |
| -H | Hidden File or Directory |
| -R | Read-only File or Directory |
| -S | System File or Directory |
| -e | File or Directory Exists |
| -f | Ordinary File |
| -o | Ownership (Same as Existence on an OS/2 FAT file system) |
| -r | Readable (Same as ordinary file on an OS/2 FAT file system) |
| -w | Writable (Not Read-only) |
| -x | Executable (Has a .csh, .exe, .com or .cmd extension and, if it's an .exe and .com file, appears to be a valid OS/2 binary executable.) |
| -z | Zero-length File |

Example:

```
if (-d $a) then
  echo $a is a directory
end
```

## Variable Substitution

Variable substitution is typically used to pass the value of a variable as an argument to a command. For example: `cl -AS -G2 -Zi $a.c`

| Reference | Meaning |
|---|---|
| **$var** ${**var**} | Value of variable **var**. |
| **$var**[<expr>] ${**var**[<expr>]} | value of **var**, indexed by an arbitrarily complex expression. |
| **$#var** ${**#var**} | Number of words in **var**. |
| **$?var** ${**?var**} | 1 if **var** exists; 0 otherwise. |
| $< | Pseudo-variable result of reading one line from stdin each time it's evaluated. |
| $0 .. $9 | Same as $argv[0] .. $argv[9]. |
| **$proc**(<exprlist>) | Substitute in the result of calling the procedure **proc**. Arguments can be given as a list of arbitrarily complex expressions. |
| $(<statement_list>) | Alternate comand substitution. Substitute in the stdout result of running the statement list inside the parenthesis in a child thread, discarding escape sequences and turning newlines into spaces. Similar to `` `...` ``-style command substitution except it's a little simpler for nesting several levels. |

## Substitution Modifiers

| Operator | Meaning |
|---|---|
| **:n** | **n**th word. |
| :^ | Word number 1, counting from 0 |
| :$ | Last word. |
| :% | Word matched by a !?**str**? history search. |
| **:n-m** | **n**th through **m**th words |
| **:-n** | 0 through **n**th words. |
| **:n-** | **n** through next-to-last words. |
| **:n*** | **n** through last word. |
| :* | 1 thru last word. |
| :q | Single quote each word. |
| :s/**str1**/**str2**/ | Substitute **str2** for **str1**. |
| :& | Repeat last substitution. |
| :g... | Global editing: apply the edit operation everywhere it matches, not just the first occurrence. |
| :x | Treat each word as a string and break it up into words. |
| :p | Print the substitution but don't execute the statement. (Ignored except in history substitutions.) |

## Pathname Editing on x\y\z.c

| Operator | Name | Meaning | Result |
|---|---|---|---|
| :h | head | Directory containing | x\y |
| :r | root | Path w/o .ext | x\y\z |
| :t | tail | Simple filename | z.c |
| :e | ext | .ext w/o the "." | c |
| :f | fullpath | Fully-qualified name | d:\bob\x\y\z.c |

Example: `echo $path:gt`

## Predefined Variables

Legend for this table:

| | |
|---|---|
| COURIER | A **setenv** environmental variable. Environmental variables are passed to any child processes or screens you create by invoking an external utility or application. When Hamilton C shell starts up it looks for the ones shown here to be defined in the environment it inherits; if they're not already defined, the shell creates them. |
| **bold** | A **set** variable shared by all threads: if one makes a change, all will see it. |
| normal | Each thread gets its own copy but the initial value is inherited from its parent. |
| *italics* | Each thread gets its own copy but the initialization is always to a defined value. |

| Name | Default | Use |
|---|---|---|
| @ | | A synonym for the stmtnumber variable. |
| < | | A synonym for the getline variable. |
| *argv* | | Any argument words passed to the shell or to a `.csh` batch file. |
| **bsdhistory** | 0 | By default, "`!!`" is the immediately preceding command and "`!-1`" is the one before that. Setting `bsdhistory` = 1 makes them the same. |
| cdhome | 0 | If set, "`cd`" with no argument is the same as "`cd $home`"; default is to simply print the current directory name. |
| cdisk | | Current disk, not including colon. |
| CDISK | | Same as cdisk, but in upper case. |
| **cdpath** | null | List of directories to search for the subdirectory specified as the new current directory. |
| chgdisk | 0 | If set, `cd` automatically does a DosSelectDisk if the path is on another disk. |
| *child* | 0 | Identification number of the last child process spawned. |
| COMSPEC | | Pathname of `cmd.exe`. |
| cwd | | Full pathname of the current directory. |
| DRIVEMASK | | Used by `du.exe`, `pwd.exe` and `vl.exe` to limit the default list of drives it will report on. Written as a list of alphabetic characters representing the drives you want listed; ranges are allowed. If you don't define this variable, all drives beginning with `C:` are normally reported. |

## Predefined Variables

| Name | Default | Use |
|---|---|---|
| *echoinput* | 0 | Copy the input to stdout as it's read. |
| *eofgetline* | 0 | Pseudo-variable to indicate if the last reference to getline encountered an end-of-file condition. |
| ESCAPESYM | ^ | Character to be interpreted as a literal escape character. |
| **escapesym** | ^ | Same as the ESCAPESYM environmental variable. |
| *getchar* | | Read one character from stdin without echoing. If stdin is tied to the keyboard, outboard keys are returned as a two-character string. |
| *getline* | | Read one line from stdin pseudo-variable. If stdin is tied to the keyboard, keystrokes are echoed as they're typed. |
| gotowindow | 50 | Number of statements a goto can jump over (when not inside a nested block) without being considered an error. |
| **histchars** | !% | Characters which introduce long-form and short-form history references, respectively. |
| *history* | 0 | Number of statements to remember on the history list; 0 turns off the history mechanism. (If the thread is interactive, history is automatically set to 100.) |
| HOME | | Home directory (default is the initial current directory.) |
| **home** | | Same as the HOME environmental variable. |
| *ignoreeof* | 0 | If True, don't exit at EOF on stdin; insist on an exit command. |
| *ignoreerrors* | 0 | Determine whether execution should continue if an error occurs: 0 means the thread exits; 1 (the default for an interactive thread) means exit from loops or procedures and try to read a new command; 2 means ignore all errors. |
| *ignorestatus* | 1 | If True, a non-zero status code from a child process is ignored. Otherwise, it's an error. |
| *interactive* | 0 | If True, prompt for input. |
| nohashing | 0 | If True, turn off hashing of the directories on the search path. |

## Predefined Variables

| Name | Default | Use |
|---|---|---|
| noclobber | 0 | If True, don't allow redirection to overwrite an existing file unless the "!" override is given. |
| noglob | | A synonym for the nowild variable. |
| nonohidden | 0 | Determine whether wildcarding will match against hidden files: 0 means don't match hidden files; 1 means hidden files will be found. |
| nonomatch | 0 | Determine the response to a wildcard that doesn't match anything: 0 means it's an error; 1 means pass it through to the application; 2 means simply discard it. |
| nonovar | 0 | Determine the response to a non-existent variable, procedure or alias. Same encoding as nonomatch. |
| nowild | 0 | If True, turn off filename wildcarding. |
| nullwords | 0 | Determines whether an array index off the end of a list is an error (0) or returns a null word (1.) |
| `PATH` | | Search path for executable files. |
| **path** | | Same as the `PATH` environmental variable, broken into words. |
| precision | 6 | Number of decimal places to print when displaying floating point values. |
| `PROMPT1` | `$@ $CDISK%` | Primary command prompt template. |
| **prompt1** | | Same as the `PROMPT1` environmental variable. |
| `PROMPT2` | `$@ $CDISK?` | Continuation line prompt template. |
| **prompt2** | | Same as the `PROMPT2` environmental variable. |
| `RADIX` | | Default radix used by `more.exe` when displaying binary data. If not defined, `RADIX` = 16 is used. |
| **savehist** | 0 | Save the history contents into `history.csh` in the home directory. |
| *scriptname* | | Name of the C shell script file being executed, if any. |
| `SHELL` | | Always set to the pathname of the Hamilton C shell `csh.exe` file. |
| **shell** | | Same as the `SHELL` environmental variable. |
| *status* | 0 | Exit code of the last child process. |

## Predefined Variables

| Name | Default | Use |
|---|---|---|
| *stmtnumber* | 1 | Autoincremented statement number used with the history list and in prompting. |
| SWITCHCHARS | | Characters that can be used as option introducers for the shell and utilities. If undefined, "-" and "/" are used. |
| TABS | 8 | Used by `more.exe` to tell it how many character positions there are between tab stops. |
| tailstatus | 0 | Determines whether the status variable will reflect the reflect the return code from the leftmost or rightmost stage of a pipeline: 0 means leftmost; 1 means rightmost. |
| *threadid* | | Thread id of the currently executing thread. |
| TZ | | Used by `tar.exe` to tell it how to convert between local time and GMT. The TZ variable should be in the form of a three-letter timezone, e.g., EST, followed by a signed number giving the difference in hours between GMT and local time, followed by an optional daylight savings timezone. Examples are EST5EDT in New York or PST8PDT in California. |
| verbose | 0 | If True, print out all available information when reporting errors. |

# Help for Hamilton C shell

## csh: Startup the Hamilton C shell

Usage: csh [-!cCefFhiLlnsZ-] [ arguments ... ]

Options:

- -! Ignore errors: Continue execution even if a command terminates abnormally. (Implied by interactive.)
- -c Execute the command following on the command line, then exit. (Implies not interactive.)
- -C Immediately execute the command on the command line, then continue with normal startup and processing of stdin. (Useful for specifying a change directory to the home directory when starting up as a new OS/2 session.)
- -e Echo the raw input to stdout.
- -f Fast startup: Don't look for a `startup.csh` file.
- -F *Faster* startup: Don't look for a `startup.csh` file and don't hash the path directories.
- -i Interactive (even if stdin appears to be a file or a pipe): Prompt for input and show the result of history substitutions.
- -L Login shell: Look for `login.csh` and `logout.csh` and do history save at exit if savehist == 1.
- -l same as -L.
- -n No execution: Parse commands looking for syntax errors but don't execute them.
- -s Read and execute a single line from stdin. (Implies not interactive.)
- -Z Very special purpose: Don't bump the maximum file handle count during shell initialization. Use this option as a workaround if you encounter an application that fails if it inherits a larger limit. This option only works from the Start Programs or Group menus, not the command line.
- -h Help.
- -- End of options.

(If preferred, the slash, "/," may be used in place of a minus to specify options to csh.exe or any of the utilities.)

## Regular Expressions

Regular expressions are used by `grep` and `sed` for text search and replace operations. They're a bit more complex than wildcards used by the shell but better suited to manipulating large text files. Regular expressions are written in this notation, in decreasing precedence:

| Characters | Meaning |
|---|---|
| c | Any ordinary character matches itself. |
| \c | Match the literal character c. |
| ^ | Beginning of line. |
| $ | End of line. |
| . | Match any single character. |
| [...] | Match any single character in the list. |
| [^...] | Match any single character not in the list. |
| \n | Match whatever literal text the n'th tagged \(...\) expression matched. |
| r* | Match zero or more occurrences of r. (In a regular expression, "*" doesn't match anything by itself; it's only a *postfix* operator against the previous expression.) |
| r1r2 | Match expression r1 followed by r2. |
| \(r\) | Tagged regular expression. Match the pattern inside the \(...\), and remember the literal text that matched. |

In addition, in a `sed` replace string, "&" refers to whatever the search string matched.

Since many regular expression characters have special meaning to the C shell, it's conventional to single quote any regular expressions on the command line. Also, type two "^" characters to mean one except when it immediately follows "[." For example, to look for the word "`main`" followed by matched parenthesis in all the `.c` files:

```
% grep -n 'main.*(.*)' *.c
cat.c:163:void main(argc, argv)
chmod.c:305:void main (argc, argv)
cut.c:871:void main(argc, argv)
date.c:109:void main(argc, argv)
:
```

## Sample Applications

**Factor.csh:** A self-loading procedure which prints a list of the factors of a number, illustrating the use of recursion.

```
proc factor(n)
  if (n > 2) then
     for i = 2 to floor(sqrt(n)) do
        if (n % i == 0) then
           echo $i
           return factor(n/i)
        end
     end
  end
  return n
end

factor $argv
```

Invoked as:

```
factor 6324489
```

It would print:

```
3
3
702721
```

To print the factors on one line and time how long it takes:

```
time echo `factor 6324489`
```

The `...` sequence means command substitution: run what's inside the backquotes and substitute the output back onto the command line. This would print:

```
3 3 702721
0:00:02.35
```

## Sample Applications

**ts:** A procedure to do a simple text search of all files with a given extension anywhere in a directory tree.

```
proc ts(startdir, ext, text)
  local files
  pushd -s $startdir
  set files = `ls -rD1 | grep -i \.$ext^'^$^'`
  if (files != '') fgrep -in "$text" $files
  popd -s
end
```

`ts` works by pushing the directory to be searched onto the directory stack, making it the current disk and directory. `ls` and `grep` are used to recursively list all files (not directories) anywhere in the tree that end with *.ext* where *ext* is whatever the caller requests.

Assuming there are some files, `fgrep` is used to search them, ignoring character case and showing line numbers of any matches. (The `if` test is needed since calling `fgrep` without a filename argument would cause it to try to read stdin.)

The text argument is inside double quotes in case the search text is more than one word. The output can of course be piped to `more`:

```
% ts ~\source c DosWrite | more
```

**duplicat:** A procedure to print the list of all filenames that appear more than once anywhere in a directory tree:

```
proc duplicat(startdir)
  local i
  foreach i (`ls -r $startdir`:gt)
     calc i
  end | sort | uniq -d
end
```

`duplicat` works by making a list of the entire contents of the directory tree using `ls`. The `:gt` operator means globally edit the list to trim each pathname down to just the tail part; e.g., given "`x\y\z.c`", the tail is just "`z.c`".

The `foreach` loop writes each name out to the pipe, one per line. The `sort` obviously sorts all the lines alphabetically and the `uniq -d` command gives just the duplicates. Here's an example run against a very full 100MB HPFS partition:

```
% time duplicat h:\ > g:duplist
0:02:08.69
```

## Sample Applications

**Whereis.csh:** A self-loading procedure to find all the files anywhere on the search path corresponding to the command name, illustrating pattern matching and file system tests.

```
proc whereis(name)
  local i, j
  if (name =~ "*.*") then
     foreach i ($path)
        if (i =~ "*\") then
           if (-e $i$name) echo $i$name
        else
           if (-e $i\$name) echo $i\$name
        end
     end
  else
     foreach i ($path)
        if (i =~ "*\") then
           foreach j (.csh .exe .com .cmd)
              if (-e $i$name$j) echo $i$name$j
           end
        else
           foreach j (.csh .exe .com .cmd)
              if (-e $i\$name$j) echo $i\$name$j
           end
        end
     end
  end
end

whereis $argv
```

Invoked as:

```
whereis ls
```

It would print:

```
c:\os2\bin\ls.exe
```

`ls.exe` is the file directory lister. Invoked as:

```
time ls -l `whereis more`
```

It would show the two versions of `more`. (Our `more` "is less filling and tastes better.")

```
---A-  Jun 19 10:00    21897  f:\os2\hamilton\more.exe
---A-  Apr 28 12:00    34881  f:\os2\ibm\more.com
0:00:01.47
```

"Indispensable. Easy to set up. Built to let experienced OS/2 users adapt with little hassle. A comfortable mix of command history and editing using the editing keys ... potent, capable ... much richer than its BSD Unix counterpart."

*-- Tom Yager, BYTE Magazine, February 1990*

"Much more powerful than CMD.EXE ... blindingly fast ... we have a winner ... a much-needed and well-done product."

*-- Kenneth G. Goutal, Personal Workstation Magazine, September 1989*

"Hamilton C shell is a killer shell -- an essential development tool. The command line editing is just what I wanted. I use Hamilton C shell throughout the day and recommend it highly."

*-- Tracy Licklider*

"A great set of software development tools. Your C shell becomes a part of OS/2 instead of being an incomplete UNIX code port that does an injustice to both operating systems. It's really nice using tools that were designed for OS/2. Any OS/2 developer would appreciate the C shell. Keep up the good work."

*-- P.M. Callihan, Duck Run Electronics*

# Hamilton C shell™

## Customer comments:

"I'm very impressed with the product. We get flack from folks because we aren't 'UNIX®-like,' when actually we are rather UNIX-like at the program interface level; it's our user interface that's very un-UNIX-like. Your product not only has its tangible benefits, but also some psychological ones as well. Personally, I'd like to have a copy for my home machine."

*-- Gordon Letwin*

"Hamilton C shell is a hacker's dream -- it makes all your other software tools work faster and better. Hamilton has given us two-inch pipe where Microsoft and IBM gave us quarter-inch; a full set of controls where they gave us an off switch; and a Porsche engine where they gave us a VW."

*-- Martin Heller, Software developer and OS/2® consultant*

"A fine product. Outstanding command line editing and a complete set of terrific utilities. No one should cripple OS/2 with little more than a DOS prompt when they can have Hamilton C shell. Buy it!"

*-- Dave Nanian, co-author of BRIEF*

"Hamilton C shell outperforms CMD.EXE in terms of usefulness and flexibility by 3 or 4 orders of magnitude and even blasts the BSD 4.3 C shell I've been using on an Apollo Domain 4000 right out of the water."

*-- Mark Montague, University of Michigan*

"I found the Hamilton C shell to satisfy two needs. It saves time and it eased my transition to an OS/2 environment. I use Hamilton C shell as a productivity tool. I like to deal directly with a developer who cares a lot. Doug is very cooperative and responsive."

*-- Wayne Chin, Hewlett-Packard*

"I use it all the time. My work requires me to use both UNIX and OS/2. Hamilton C shell made learning OS/2 bearable and makes my work easier and more enjoyable. It's one of the best products I've used. Am I getting my money's worth? Yes!!"

*-- Mark Sontz, Language Technology*

**Hamilton Laboratories,** 13 Old Farm Road, Wayland, MA 01778, 508-358-5715