BUILTINS SHELL A RECODER

cd [-L|-P] [dir]

Change the current directory to dir.

The variable **HOME** is the default dir.

The variable **CDPATH** defines the search path for the directory containing dir.

Alternative directory names in **CDPATH** are separated by a colon (**:**).

A null directory name in **CDPATH** is the same as the current directory, *i.e., “***.***”*.

If dir begins with a slash (*/*), then **CDPATH** is not used.

The **-P** option says to use the physical directory structure instead of following symbolic links (see also the -P option to the set builtin command); the **-L** option forces symbolic links to be followed.

An argument of **-** is equivalent to **$OLDPWD**. If a non-empty directory name from **CDPATH** is used, or if **-** is the first argument, and the directory change is successful, the absolute pathname of the new working directory is written to the standard output.

The return value is true if the directory was successfully changed; false otherwise.

echo [-neE] [arg ...]

Output the args, separated by spaces, followed by a newline.

The return status is always 0.

If **-n** is specified, the trailing newline is suppressed.

If the **-e** option is given, interpretation of the following backslash-escaped characters is enabled.

The **-E** option disables the interpretation of these escape characters, even on systems where they are interpreted by default.

The xpg_echo shell option may be used to dynamically determine whether or not echo expands these escape characters by default.

echo does not interpret **--** to mean the end of options.

echo interprets the following escape sequences:

\a	alert (bell)
\b	backspace
\c	suppress trailing newline
\e	an escape character
\f	form feed
\n	new line
\r	carriage return

\t	horizontal tab
\v	vertical tab
\\	backslash
\0nnn	the eight-bit character whose value is the octal value nnn (0 to 3 octal digits)
\xHH	the eight-bit character whose value is the hexa value HH (1 or 2 hex digits)

exit [n]

Cause the shell to exit with a status of n.

If **n** is omitted, the exit status is that of the last command executed.

A trap on **EXIT** is executed before the shell terminates.

export [-fn] [name[=word]] ...
export -p

The supplied names are marked for automatic export to the environment of subsequently executed commands.

If the **-f** option is given, the names refer to functions.

If no names are given, or if the **-p** option is supplied, a list of all names that are exported in this shell is printed.

The **-n** option causes the export property to be removed from each name.

If a variable **name** is followed by **=word**, the value of the variable is set to word.

export returns an exit status of 0 unless an invalid option is encountered, one of the names is not a valid shell variable name, or **-f** is supplied with a name that is not a function.

pwd [-LP]

Print the absolute pathname of the current working directory.

The pathname printed contains no symbolic links if the **-P** option is supplied or the **-o** physical option to the ***set builtin*** command is enabled.

If the **-L** option is used, the pathname printed may contain symbolic links.

The return status is 0 unless an error occurs while reading the name of the current directory or an invalid option is supplied.

unset [-fv] [name ...]

For each name, remove the corresponding variable or function.

If **no options** are supplied, or the **-v** option is given, each name refers to a shell variable.

Read-only variables may not be unset.

If **-f** is specified, each name refers to a shell function, and the function definition is removed.

Each **unset** variable or function is removed from the environment passed to subsequent commands.

If any of **RANDOM**, **SECONDS**, **LINENO**, **HISTCMD**, **FUNCNAME**, **GROUPS**, or **DIRSTACK** are **unset**, they lose their special properties, even if they are subsequently reset.

The exit status is true unless a name is readonly.

Infos accessibles depuis man bash or man unset

env [-iv] [-P altpath] [-S string] [-u name]
[name=value ...]
[utility [argument ...]]

env -- set environment and execute command, or print environment

The env utility executes another utility after modifying the environment as specified on the command line.

Each **name=value** option **specifies the setting of an environment variable**, name, with a value of value.

All such environment variables are set before the utility is executed.

The options are as follows:

-i Execute the utility with only those environment variables specified by name=value options. *The environment inherited by env is ignored completely*.

-P altpath

Search the set of directories as specified by altpath to locate the specified utility program, instead of using the value of the **PATH** environment variable.

-S string

Split apart the given string into multiple strings, and process each of the resulting strings as separate arguments to the env utility. The **-S** option recognizes some special character escape sequences and also supports environment-variable substitution, *as described below*.

-u name

If the environment variable name is in the environment, then remove it before processing the remaining options. *This is similar to the unset command in sh(1)*. The value for name must not include the `=' character.

-v Print verbose information for each step of processing done by the env utility. Additional information will be printed if **-v** is specified multiple times.

The above options are only recognized when they are specified before any name=value options.

If no utility is specified, **env prints out the names and values of the variables in the environment, with one name/value pair per line**.

*Details of **-S** (split-string) processing*

The processing of the **-S** option will split the given string into separate arguments based on any space or <tab> characters found in the string.

Each of those new arguments will then be treated as if it had been specified as a separate argument on the original env command.

Spaces and tabs may be embedded in one of those new arguments by using single (``'') or double (`"') quotes, or backslashes (`\').

Single quotes will escape all non-single quote characters, up to the matching single quote.

Double quotes will escape all non-double quote characters, up to the matching double quote.

It is an error if the end of the string is reached before the matching quote character. If **-S** would create a new argument that starts with the `**#'** character, then that argument and the remainder of the string will be ignored. The `\#' sequence can be used when you want a new argument to start with a `#' character, without causing the remainder of the string to be skipped.

While processing the string value, **-S** processing will treat certain character combinations as escape sequences which represent some action to take. The character escape sequences are in backslash notation.

The characters and their meanings are as follows:

\c Ignore the remaining characters in the string. This must not appear inside a double-quoted string.

\f Replace with a <form-feed> character.

\n Replace with a <new-line> character.

\r Replace with a <carriage return> character.

\t Replace with a <tab> character.

\v Replace with a <vertical tab> character.

\# Replace with a `#' character. This would be useful when you need a `#' as the first character in one of the argument created by splitting apart the given string.

\$ Replace with a `$' character.

_ If this is found inside of a double-quoted string, then replace it with a single blank. If this is found outside of a quoted string, then treat this as the separator character between new arguments in the original string.

\" Replace with a <double quote> character.

\' Replace with a <single quote> character.

\\ Replace with a backslash character.

The sequences for <single-quote> and backslash are the only sequences which are recognized inside of a single-quoted string. The other sequences have no special meaning inside a single-quoted string.

All escape sequences are recognized inside of a double-quoted string.

It is an error if a single `\' character is followed by a character other than the ones listed above.

The processing of **-S** also supports substitution of values from environment variables. To do this, the name of the environment variable must be inside of `**${}**', such as: **${SOMEVAR}**. *The common shell syntax of $SOMEVAR is not supported*.

All values substituted will be the values of the environment variables as they were when the **env utility** was originally invoked. Those values will not be checked for any of the escape sequences as described above. And any settings of **name=value** will not affect the values used for substitution in **-S** processing.

Also, **-S** processing can not reference the value of the special parameters which are defined by most shells. For instance, **-S** can not recognize special parameters such as: `**$***', `**$@**', `**$#**', `**$?**' or `**$$**' if they appear inside the given string.

Use in shell-scripts the env utility is often used as the interpreter on the first line of interpreted scripts, as described in ***execve(2)***. Note that the way the kernel parses the `#!' (first line) of an interpreted script has changed as of FreeBSD 6.0. Prior to that, the FreeBSD kernel would split that first line into separate arguments based on any whitespace (space or <tab> characters) found in the line.

So, if a script named **/usr/local/bin/someport** had a first line of:

```
#!/usr/local/bin/php -n -q -dsafe_mode=0
```

then the **/usr/local/bin/php** program would have been started with the arguments of:

```
arg[0] = '/usr/local/bin/php'
arg[1] = '-n'
arg[2] = '-q'
arg[3] = '-dsafe_mode=0'
arg[4] = '/usr/local/bin/someport'
```

plus any arguments the user specified when executing someport.

However, this processing of multiple options on the `**#!**' line is not the way any other operating system parses the first line of an interpreted script.

So after a change which was made for FreeBSD 6.0 release, that script will result in **/usr/local/bin/php** being started with the arguments of:

```
arg[0] = '/usr/local/bin/php'
arg[1] = '-n -q -dsafe_mode=0'
arg[2] = '/usr/local/bin/someport'
```

plus any arguments the user specified.

This caused a significant change in the behavior of a few scripts. In the case of above script, to have it behave the same way under FreeBSD 6.0 as it did under earlier releases, the first line should be changed to:

```
#!/usr/bin/env -S /usr/local/bin/php -n -q -dsafe_mode=0
```

The env utility will be started with the entire line as a single argument:

```
arg[1] = '-S /usr/local/bin/php -n -q -dsafe_mode=0'
```

and then **-S** processing will split that line into separate arguments before executing **/usr/local/bin/php**.

The **env utility** uses the **PATH** environment variable to locate the requested utility if the name contains **no** `**/**' characters, unless he **-P** option has been specified.

RETURN

The **env utility exits 0 on success**, and **> 0 if an error occurs**.

An exit status of **126** indicates that utility was found, but could not be executed.

An exit status of **127** indicates that utility could not be found.

EXAMPLES

Since the env utility is often used as part of the first line of an interpreted script, the following examples show a number of ways that the env utility can be useful in scripts.

The kernel processing of an interpreted script does not allow a script to directly reference some other script as its own interpreter. As a way around this, the main difference between

```
#!/usr/local/bin/foo
```

and

```
#!/usr/bin/env /usr/local/bin/foo
```

is that the latter works even if **/usr/local/bin/foo** is itself an interpreted script.

Probably the most common use of env is to find the correct interpreter for a script, when the interpreter may be in different directories on different systems.

The following example will find the `perl' interpreter by searching through the directories specified by PATH.

```
#!/usr/bin/env perl
```

One limitation of that example is that it assumes the user's value for PATH is set to a value which will find the interpreter you want to execute. The -P option can be used to make sure a specific list of directories is used in the search for utility. Note that the -S option is also required for this example to work correctly.

```
#!/usr/bin/env -S -P /usr/local/bin:/usr/bin perl
```

The above finds `perl' only if it is in /**usr/local/bin** or **/usr/bin**. That could be combined with the present value of **PATH**, to provide more flexibility. Note that spaces are not required between the -S and -P options:

```
#!/usr/bin/env -S-P/usr/local/bin:/usr/bin:${PATH} perl
```

COMPATIBILITY

The env utility accepts the - option as a synonym for -i.

BUGS

The env utility does not handle values of utility which have an equals sign (`=') in their name, for obvious reasons.

The env utility does not take multibyte characters into account when processing the -S option, which may lead to incorrect results in some locales.

man env

https://docs.google.com/document/d/1cVZLCoBR7vZ5CPXHqgbl77E5Sx_v3CxwF5FeIwqVpJw/edit