## NAME

Shell - run shell commands transparently within perl

## SYNOPSIS

```
use Shell qw(cat ps cp);
$passwd = cat('</etc/passwd');
@pslines = ps('-ww'),
cp("/etc/passwd", "/tmp/passwd");

# object oriented
my $sh = Shell->new;
print $sh->ls('-l');
```

## DESCRIPTION

### Caveats

This package is included as a show case, illustrating a few Perl features. It shouldn't be used for production programs. Although it does provide a simple interface for obtaining the standard output of arbitrary commands, there may be better ways of achieving what you need.

Running shell commands while obtaining standard output can be done with the `qx/STRING/` operator, or by calling `open` with a filename expression that ends with `|`, giving you the option to process one line at a time. If you don't need to process standard output at all, you might use `system` (in preference of doing a print with the collected standard output).

Since Shell.pm and all of the aforementioned techniques use your system's shell to call some local command, none of them is portable across different systems. Note, however, that there are several built in functions and library packages providing portable implementations of functions operating on files, such as: `glob`, `link` and `unlink`, `mkdir` and `rmdir`, `rename`, `File::Compare`, `File::Copy`, `File::Find` etc.

Using Shell.pm while importing `foo` creates a subroutine `foo` in the namespace of the importing package. Calling `foo` with arguments `arg1`, `arg2`,... results in a shell command `foo arg1 arg2...`, where the function name and the arguments are joined with a blank. (See the subsection on Escaping magic characters.) Since the result is essentially a command line to be passed to the shell, your notion of arguments to the Perl function is not necessarily identical to what the shell treats as a command line token, to be passed as an individual argument to the program. Furthermore, note that this implies that `foo` is callable by file name only, which frequently depends on the setting of the program's environment.

Creating a Shell object gives you the opportunity to call any command in the usual OO notation without requiring you to announce it in the `use Shell` statement. Don't assume any additional semantics being associated with a Shell object: in no way is it similar to a shell process with its environment or current working directory or any other setting.

### Escaping Magic Characters

It is, in general, impossible to take care of quoting the shell's magic characters. For some obscure reason, however, Shell.pm quotes apostrophes (`'`) and backslashes (`\`) on UNIX, and spaces and quotes (") on Windows.

### Configuration

If you set $Shell::capture_stderr to true, the module will attempt to capture the standard error output of the process as well. This is done by adding `2>&1` to the command line, so don't try this on a system not supporting this redirection.

If you set $Shell::raw to true no quoting whatsoever is done.

## BUGS

Quoting should be off by default.

It isn't possible to call shell built in commands, but it can be done by using a workaround, e.g. shell( '-c', 'set' ).

Capturing standard error does not work on some systems (e.g. VMS).

## AUTHOR

```
Date: Thu, 22 Sep 94 16:18:16 -0700
Message-Id: <9409222318.AA17072@scalpel.netlabs.com>
To: perl5-porters@isu.edu
From: Larry Wall <lwall@scalpel.netlabs.com>
Subject: a new module I just wrote
```

Here's one that'll whack your mind a little out.

```
#!/usr/bin/perl

use Shell;

$foo = echo("howdy", "<funny>", "world");
print $foo;

$passwd = cat("</etc/passwd");
print $passwd;

sub ps;
print ps -ww;

cp("/etc/passwd", "/etc/passwd.orig");
```

That's maybe too gonzo. It actually exports an AUTOLOAD to the current package (and uncovered a bug in Beta 3, by the way). Maybe the usual usage should be

```
use Shell qw(echo cat ps cp);
```

Larry Wall

Changes by Jenda@Krynicky.cz and Dave Cottle <d.cottle@csc.canterbury.ac.nz>.

Changes for OO syntax and bug fixes by Casey West <casey@geeknest.com>.

`$Shell::raw` and pod rewrite by Wolfgang Laun.