How NOT to capture output in Perl

David Golden
dagolden@cpan.org
Spring 2009

"How do I capture output in Perl?"

Simple question

Deceptively simple

So what's easy?

STDOUT
of a command

Backticks

```
$out = `perl -E 'say "Hello World"'`;
```

Digression...

In real life, use $^X not perl

```
$out = `$^X -E 'say "Hello World"'`;
$out = `perl -E 'say "Hello World"'`;
```

Digression...

$^X = current perl

'perl' = first in PATH

```
$out = `$^X -E 'say "Hello World"'`;
$out = `perl -E 'say "Hello World"'`;
```

piped open()

```
open PIPE, qq{perl -E 'say "Hello World"' |};
$out = do { local $/; <PIPE> };
```

Backticks and open use the shell

Different shells
Different quotes

/bin/sh
vs.
cmd.exe

```
$out = `perl -E 'say "Hello World"'`;
$out = `perl -E "say qq{Hello World}"`;
```

Filenames can contain spaces

```
$file = "Hello World.txt";
$out = `perl -E 'say shift' $file`;
```

How the shell sees it

```
$ perl -E 'say shift' Hello World.txt
Hello
```

Need backticks like system(@cmd)

IPC::System::Simple

```
use IPC::System::Simple 'capture';
@cmd = ('perl', '-E', 'say shift', 'Hello World.txt');
$out = capture(@cmd);
```

Capturing STDERR?

perlfaq –q capture

How can I capture STDERR from an external command?

There are three basic ways of running external commands:

```
  system $cmd;            # using system()
  $output = `$cmd`;       # using backticks (``)
  open (PIPE, "cmd |");   # using open()
```

With system(), both STDOUT and STDERR will go the same place as the script's STDOUT and STDERR, unless the system() command redirects them. Backticks and open() read only the STDOUT of your command.

You can also use the open3() function from IPC::Open3. Benjamin Goldberg provides some sample code:

To capture a program's STDOUT, but discard its STDERR:

```
  use IPC::Open3;
  use File::Spec;
  use Symbol qw(gensym);
  open(NULL, ">", File::Spec->devnull);
  my $pid = open3(gensym, \*PH, ">&NULL", "cmd");
  while( <PH> ) { }
  waitpid($pid, 0);
```

To capture a program's STDERR, but discard its STDOUT:

```
  use IPC::Open3;
  use File::Spec;
  use Symbol qw(gensym);
  open(NULL, ">", File::Spec->devnull);
  my $pid = open3(gensym, ">&NULL", \*PH, "cmd");
  while( <PH> ) { }
  waitpid($pid, 0);
```

To capture a program's STDERR, and let its STDOUT go to our own STDERR:

```
  use IPC::Open3;
  use Symbol qw(gensym);
  my $pid = open3(gensym, ">&STDERR", \*PH, "cmd");
  while( <PH> ) { }
  waitpid($pid, 0);
```

To read both a command's STDOUT and its STDERR separately, you can redirect them to temp files, let the command run, then read the temp files:

```
  use IPC::Open3;
  use Symbol qw(gensym);
  use IO::File;
  local *CATCHOUT = IO::File->new_tmpfile;
  local *CATCHERR = IO::File->new_tmpfile;
  my $pid = open3(gensym, ">&CATCHOUT", ">&CATCHERR", "cmd");
  waitpid($pid, 0);
  seek $_, 0, 0 for \*CATCHOUT, \*CATCHERR;
  while( <CATCHOUT> ) {}
  while( <CATCHERR> ) {}
```

But there's no real need for *both* to be tempfiles... the following should work just as well, without deadlocking:

```
  use IPC::Open3;
  use Symbol qw(gensym);
  use IO::File;
  local *CATCHERR = IO::File->new_tmpfile;
  my $pid = open3(gensym, \*CATCHOUT, ">&CATCHERR", "cmd");
  while( <CATCHOUT> ) {}
  waitpid($pid, 0);
  seek CATCHERR, 0, 0;
  while( <CATCHERR> ) {}
```

And it'll be faster, too, since we can begin processing the program's stdout immediately, rather than waiting for the program to finish.

With any of these, you can change file descriptors before the call:

```
  open(STDOUT, ">logfile");
  system("ls");
```

or you can use Bourne shell file-descriptor redirection:

```
  $output = `$cmd 2>some_file`;
  open (PIPE, "cmd 2>some_file |");
```

You can also use file-descriptor redirection to make STDERR a duplicate of STDOUT:

```
  $output = `$cmd 2>&1`;
  open (PIPE, "cmd 2>&1 |");
```

Note that you cannot simply open STDERR to be a dup of STDOUT in your Perl program and avoid calling the shell to do the redirection. This doesn't work:

```
  open(STDERR, ">&STDOUT");
  $alloutput = `cmd args`;  # stderr still escapes
```

This fails because the open() makes STDERR go to where STDOUT was going at the time of the open(). The backticks then make STDOUT go to a string, but don't change STDERR (which still goes to the old STDOUT).

Note that you must use Bourne shell (sh(1)) redirection syntax in backticks, not csh(1)! Details on why Perl's system() and backtick and pipe opens all use the Bourne shell are in the versus/csh.whynot article in the "Far More Than You Ever Wanted To Know" collection in http://www.cpan.org/misc/olddoc/FMTEYEWTK.tgz
To capture a command's STDERR and STDOUT together:

```
  $output = `cmd 2>&1`;                       # either with backticks
  $pid = open(PH, "cmd 2>&1 |");              # or with an open pipe
  while (<PH>) { }                            #    plus a read
```

To capture a command's STDOUT but discard its STDERR:

```
  $output = `cmd 2>/dev/null`;                # either with backticks
  $pid = open(PH, "cmd 2>/dev/null |");       # or with an open pipe
  while (<PH>) { }                            #    plus a read
```

To capture a command's STDERR but discard its STDOUT:

```
  $output = `cmd 2>&1 1>/dev/null`;           # either with backticks
  $pid = open(PH, "cmd 2>&1 1>/dev/null |");  # or with an open pipe
  while (<PH>) { }                            #    plus a read
```

To exchange a command's STDOUT and STDERR in order to capture the STDERR but leave its STDOUT to come out our old STDERR:

```
  $output = `cmd 3>&1 1>&2 2>&3 3>&-`;        # either with backticks
  $pid = open(PH, "cmd 3>&1 1>&2 2>&3 3>&-|");# or with an open pipe
  while (<PH>) { }                            #    plus a read
```

To read both a command's STDOUT and its STDERR separately, it's easiest to redirect them separately to files, and then read from those files when the program is done:

```
  system("program args 1>program.stdout 2>program.stderr");
```

Ordering is important in all these examples. That's because the shell processes file descriptor redirections in strictly left to right order.

```
  system("prog args 1>tmpfile 2>&1");
  system("prog args 2>&1 1>tmpfile");
```

The first command sends both standard out and standard error to the temporary file. The second command sends only the old standard output there, and the old standard error shows up on the old standard out.

Screw all that!*

* Actually, it's good reading on the fundamentals if you want to reinvent the wheel

CPAN, help!

IPC::Run3

```
use IPC::Run3 'run3';
run3(\@cmd, undef, \$out, \$err);
```

Variations

```
use IPC::Run3 'run3';
run3(\@cmd, \&input, \$out, \@err);
```

Capturing STDOUT & STDERR together?

Sure!

```
use IPC::Run3 'run3';
run3(\@cmd, undef, \$both, \$both);
```

What's the catch?

User interaction

prompt.pl

```
#!/usr/bin/perl

while( 1 ) {
   print "Type 'exit' to quit:\n"
   $line = <STDIN>;
   last if $line =~ /^exit$/;
   print "You said: $_";
}
```

Prompts are captured

```
# try-run3.pl
use IPC::Run3 'run3';
print "Running prompt.pl\n";
run3('perl prompt.pl', undef, \$out, \$err);
```

Hangs

```
$ perl try-run3.pl
Running prompt.pl
```

(... nothing else happens ... eventually hit CTRL-C ...)

Need to capture
and echo to screen

Tee?

```
system( "$cmd | tee output.log" );
$out = do {
  local (@ARGV,$/) = "output.log"; <>
};
```

If 'tee' exists*

* Tee.pm provides 'ptee'

Gives wrong exit value

```
system( "$cmd | tee output.log" );
if ($? == 0) { print "tee ran ok" };
```

IPC::Open3

```
use IPC::Open3;
my $pid = open3(
  \*CHILD_IN, \*CHILD_OUT, \*CHILD_ERR, @cmd
);
```

select() between output handles

```
# Adapted and simplified from IPC::Open3::Simple
use IPC::Open3;
use IO::Select;
my ($out, $err) = ( '', '' );
my $pid =
  open3(\*CHILD_IN, \*CHILD_OUT, \*CHILD_ERR, @cmd);
my $reader = IO::Select->new(\*CHILD_OUT, \*CHILD_ERR);
while ( my @ready = $reader->can_read() ) {
  foreach my $fh (@ready) {
    my $line = <$fh>;
    if (!defined $line) {
      $reader->remove($fh);
      $fh->close();
    } else {
      if    (fileno($fh) == fileno(\*CHILD_OUT)) {
        $out .= $line;
        print STDOUT $line;
      }
      elsif (fileno($fh) == fileno(\*CHILD_ERR)) {
        $err .= $line
        print STDERR $line;
      }
    }
  }
}
waitpid($pid, 0);
```

But no select() on handles for Win32

IPC::Run

Win32 support
is 'experimental'

select() on sockets instead of filehandles

Still have to write interactivity code yourself

Does anything just DWIW?

IPC::Cmd

```
use IPC::Cmd 'run';
my ( $success, $error_code,
     $full_buf, $stdout_buf, $stderr_buf
   ) = run( command => $cmd, verbose => 1 );
```

In Perl core
in 5.10

Uses IPC::Run
or IPC::Open3

Interactive
plus capture

Or maybe not

```
if ( IPC::Cmd->can_capture_buffer ) {
  @results = run(@stuff);
}
else {
  die "Now what?!";
}
```

Capturing Perl?

Tied filehandles

```
tie *STDOUT, $class;
print "This string captured by $class";
```

IO::Capture
IO::Tee
Filter::Handle
Test::Output
Tie::STDOUT
Tie::STDERR

XS (or C) fails

```
use Inline C => <<'END_C';
void greet() { printf( "Tie won't capture this\n" ); }
END_C

greet;
```

system() fails

```
use Test::Output;
stdout_is { system "echo 'Hello World'" }
  "Hello World\n", "Got Hello World"; # Not OK
```

IO::CaptureOutput

```
use IO::CaptureOutput 'capture';
capture {
  print "Hello World\n"
} => \$out, \$err;
```

Perl, XS or system()

```
use IO::CaptureOutput 'capture';
capture {
   system "echo 'Hello World'"
} => \$out, \$err;
```

STDOUT & STDERR

```
use IO::CaptureOutput 'capture';
capture {
   system "echo 'Hello World'"
} => \$both, \$both;
```

Portable, too

Interactive?

No

```
use IO::CaptureOutput 'capture';
capture {
   system "perl prompt.pl" # hangs
} => \$both, \$both;
```

File::Tee

```
use File::Tee 'tee);'
tee(STDOUT, '>', 'stdout.txt');
print "Hello World\n";
system "perl prompt.pl";
# read back stdout.txt on your own
```

But uses piped open

Not portable

```
Excerpt from 'perldoc perlport':

open to "|-" and "-|" are unsupported. (Mac OS,
Win32, RISC OS)
```

Now what?

"How do I capture output in Perl?"

"How do I capture output in Perl?"

... from STDOUT, STDERR or both

"How do I capture output in Perl?"

... from STDOUT, STDERR or both
... from Perl, XS or a command

"How do I capture output in Perl?"

... from STDOUT, STDERR or both
... from Perl, XS or a command
... interactive with the terminal

"How do I capture output in Perl?"

... from STDOUT, STDERR or both
... from Perl, XS or a command
... interactive with the terminal
... that is reasonably portable

"How do I capture output in Perl?"

... from STDOUT, STDERR or both
... from Perl, XS or a command
... interactive with the terminal
... that is reasonably portable
... with a single module

Didn't exist

So I wrote it

Capture::Tiny

```
use Capture::Tiny 'capture';
$out = capture {
  print "Hello World\n";
};
```

STDOUT & STDERR

```
use Capture::Tiny 'capture';
($out, $err) = capture {
  print "Hello World\n";
  print STDERR "Goodbye World\n";
};
```

Both

```
use Capture::Tiny 'capture_merged';
$both = capture_merged {
   print "Hello World\n";
   print STDERR "Goodbye World\n";
};
```

XS or command

```
use Capture::Tiny 'capture';
($out, $err) = capture {
  system "echo 'Hello World'";
};
```

Interactive

```
use Capture::Tiny 'tee';
($out, $err) = tee {
  system "perl prompt.pl";
};
```

Fairly portable

CPAN Testers Matrix: Capture-Tiny 0.05 (latest

Distribution (e.g. DBI, CPAN-Reporter, YAML-Syck): Submit Query

CPAN User ID: Submit Query

You can click on the matrix cells or row/column headers to get the list of corresponding reports.
Alternative color schemes are available: try *View* > *Page Style* or *View* > *Use Style* in your browser.

ALL	MSWin32	darwin	freebsd	irix	linux	netbsd	openbsd	solaris
5.10.0								
5.8.9								
5.8.8								
5.8.6								
5.6.2								

Does need 'fork' except on Win32/OS2

Functional, not OO or start/stop/gather

```
@results =
  map { scalar capture { system "$cmd $_" } } @args;
```

Small

	SLOC
IPC::Run	3,522
IPC::Run3	654
IPC::Cmd	421
IPC::Open3	187
File::Tee	215
IO::CaptureOutput	181
Capture::Tiny	163

A single tool
that works

"How do I capture output in Perl?"

use Capture::Tiny

Questions?