Ptyprocess Documentation

Release 0.7

Thomas Kluyver

Feb 15, 2021

Contents

Launch a subprocess in a pseudo terminal (pty), and interact with both the process and its pty.

Sometimes, piping stdin and stdout is not enough. There might be a password prompt that doesn't read from stdin, output that changes when it's going to a pipe rather than a terminal, or curses-style interfaces that rely on a terminal. If you need to automate these things, running the process in a pseudo terminal (pty) is the answer.

Interface:

```
p = PtyProcessUnicode.spawn(['python'])
p.read(20)
p.write('6+6\n')
p.read(20)
```

Contents:

CHAPTER 1

Ptyprocess API

class `ptyprocess.`**`PtyProcess`**(*pid*, *fd*)

This class represents a process running in a pseudoterminal.

The main constructor is the `spawn()` classmethod.

classmethod spawn(*argv*, *cwd=None*, *env=None*, *echo=True*, *preexec_fn=None*, *dimensions=(24, 80)*, *pass_fds=()*)

Start the given command in a child process in a pseudo terminal.

This does all the fork/exec type of stuff for a pty, and returns an instance of PtyProcess.

If preexec_fn is supplied, it will be called with no arguments in the child process before exec-ing the specified command. It may, for instance, set signal handlers to SIG_DFL or SIG_IGN.

Dimensions of the psuedoterminal used for the subprocess can be specified as a tuple (rows, cols), or the default (24, 80) will be used.

By default, all file descriptors except 0, 1 and 2 are closed. This behavior can be overridden with pass_fds, a list of file descriptors to keep open between the parent and the child.

class `ptyprocess.`**`PtyProcessUnicode`**(*pid*, *fd*, *encoding='utf-8'*, *codec_errors='strict'*)

Unicode wrapper around a process running in a pseudoterminal.

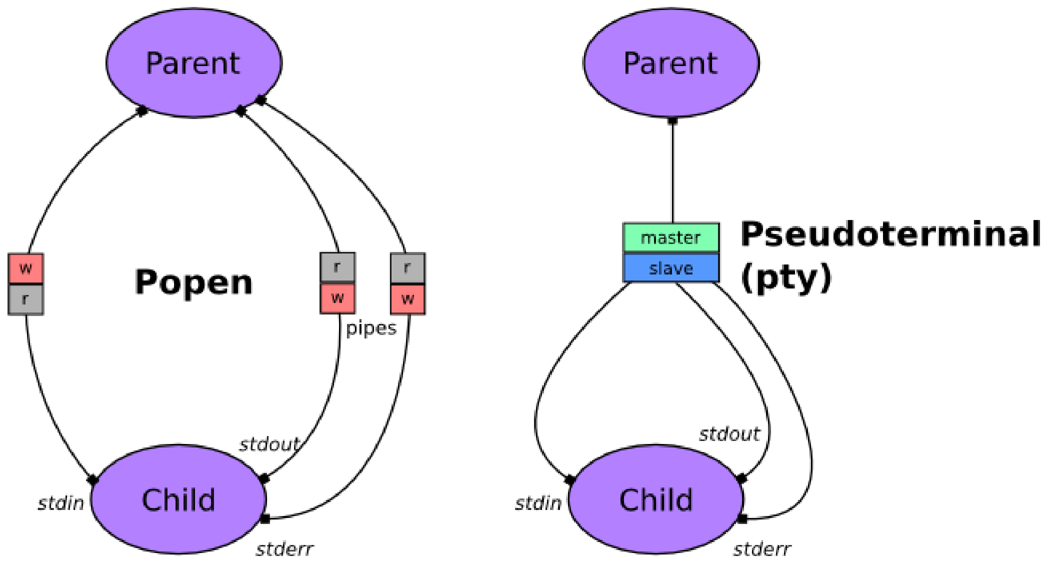
This class exposes a similar interface to `PtyProcess`, but its read methods return unicode, and its `write()` accepts unicode.

CHAPTER 2

What is a pty?

A pty is a kernel-level object which processes can write data to and read data from, a bit like a pipe.

Unlike a pipe, data moves through a single pty in both directions. When you use a program in a shell pipeline, or with `subprocess.Popen` in Python, up to three pipes are created for the process's standard streams (stdin, stdout and stderr). When you run a program using ptyprocess, all three of its standard streams are connected to a single pty:



A pty also does more than a pipe. It keeps track of the window size (rows and columns of characters) and notifies child processes (with a SIGWINCH signal) when it changes. In *cooked mode*, it does some processing of data sent from the parent process, so for instance the byte `03` (entered as Ctrl-C) will cause SIGINT to be sent to the child process.

Many command line programs behave differently if they detect that stdin or stdout is connected to a terminal instead of a pipe (using isatty()), because this normally means that they're being used interactively by a human user. They may format output differently (e.g. `ls` lists files in columns) or prompt the user to confirm actions. When you run these programs in ptyprocess, they will exhibit their 'interactive' behaviour, instead of the 'pipe' behaviour you'll see using `Popen()`.

See also:

The TTY demystified Detailed article by Linus Akesson

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index