iTerm2 Tools Documentation

Release 2.2

Aaron Meurer

July 18, 2016

Contents

iTerm2 tools for Python

Some tools for working with iTerm2's proprietary escape codes in Python.

Supports Python 2.7, 3.3, and 3.4.

The source code is on GitHub.

CHAPTER 1

Installation

```
pip install iterm2_tools
```

or

```
conda install -c asmeurer iterm2_tools
```

CHAPTER 2

Documentation

Contents:

2.1 Images

2.1.1 Functions

Functions for displaying images inline in iTerm2.

See https://iterm2.com/images.html.

iterm2_tools.images.**display_image_bytes**(*b*, *filename=None*, *inline=1*)
Display the image given by the bytes b in the terminal.

If filename=None the filename defaults to "Unnamed file".

iterm2_tools.images.**display_image_file**(*fn*)
Display an image in the terminal.

A newline is not printed.

iterm2_tools.images.**image_bytes**(*b*, *filename=None*, *inline=1*)
DEPRECATED: Use display_image_bytes.

2.1.2 Shell sequences

images.**IMAGE_CODE = '\x1b]1337;File=name={name};inline={inline};size={size}:{base64_img}\x07'**

2.2 Shell Integration

2.2.1 Functions

Shell integration

See https://groups.google.com/d/msg/iterm2-discuss/URKCBtS0228/rs5Ive4PCAAJ for documentation on the sequences, https://github.com/gnachman/iterm2-website/tree/master/source/misc for example implementations, and https://iterm2.com/shell_integration.html for a list of what this lets you do in iTerm2.

Usage

Say you have a basic REPL like:

```
input> run-command
command output
```

where `input>` is the prompt, `run-command` is the command typed by the user, and `command output` is the output of `run-command`. The basic REPL (in Python 3), would be:

```
while True:
    before_prompt()
    print("input> ", end='')
    after_prompt()
    command = input()
    before_output()
    return_val = run_command(command)
    after_output(return_val)
```

(here `return_val` should be in the range 0-255).

Note that it is recommended to use the functions (like `before_prompt()`) or the context managers (like `with Prompt()`) rather than the variables (like `BEFORE_PROMPT`) directly. These print the codes directly to stdout, avoiding potential issues with character counting.

It may be preferable to use the context managers rather than the functions, in which case, the REPL would be:

```
while True:
    with Prompt():
        print("input> ", end='')
    command = input() # raw_input() in Python 2
    with Output() as o:
        return_val = run_command(command)
        o.set_command_status(return_val)
```

However, in many cases, it is impossible to run functions before and after the prompt, e.g., when the prompt text is passed to `(raw_)input()` directly. In that case, you should use the codes directly, wrapped with `readline_invisible()`, like:

```
while True:
    command = input(
        readline_invisible(BEFORE_PROMPT) +
        "input> " +
        readline_invisible(AFTER_PROMPT
    ) # raw_input() in Python 2
    with Output() as o:
        return_val = run_command(command)
        o.set_command_status(return_val)
```

Using `readline_invisible()` is important as it tells readline to not count the codes as visible text. Without this, readline's editing and history commands will truncate text.

Notes about iTerm2:

- iTerm2 assumes that the prompt sequences will be presented in a reasonable way. Using the context managers should prevent most issues.
- The text that comes after the prompt before the first newline is read as a command. If there is no command, or the command is just whitespace, the output is effectively ignored (the same as if two before/after prompt sequences were performed without any output sequence).

- iTerm2 does not support capturing multiline commands, although the output won't include any part of the command if `before_output()` is used correctly.
- iTerm2 expects there to be nothing between `AFTER_OUTPUT` and `BEFORE_PROMPT`, except possibly more shell sequences. At the time of this writing, iTerm2's "Select Output of Last Command" actually selects the text between `BEFORE_OUTPUT` and `BEFORE_PROMPT`, not `BEFORE_OUTPUT` and `AFTER_OUTPUT` as one would expect.
- Multiline prompts are supported just fine, although the arrow will always be presented on the first line. It is not recommended to attempt to change this by not including part of the prompt between the prompt sequences (see the previous bullet point).

class `iterm2_tools.shell_integration.`**`Output`**
iTerm2 shell integration output context manager

Use like:

```
with Output() as o:
    print("output")
    o.set_command_status(status)
```

The command status should be in the range 0-255. The default status is 0.

`iterm2_tools.shell_integration.`**`Prompt`**`()`
iTerm2 shell integration prompt context manager

Use like:

```
with Prompt():
    print("Prompt:", end='')
```

`iterm2_tools.shell_integration.`**`after_output`**`(`*command_status*`)`
Shell sequence to be run after the command output.

The `command_status` should be in the range 0-255.

`iterm2_tools.shell_integration.`**`after_prompt`**`()`
Shell sequence to be run after the prompt.

`iterm2_tools.shell_integration.`**`before_output`**`()`
Shell sequence to be run before the command output.

`iterm2_tools.shell_integration.`**`before_prompt`**`()`
Shell sequence to be run before the prompt.

`iterm2_tools.shell_integration.`**`readline_invisible`**`(`*code*`)`
Wrap `code` with the special characters to tell readline that it is invisible.

2.2.2 Shell sequences

The "FinalTerm" shell sequences

`shell_integration.`**`BEFORE_PROMPT = '\x1b]133;A\x07'`**

`shell_integration.`**`AFTER_PROMPT = '\x1b]133;B\x07'`**

`shell_integration.`**`BEFORE_OUTPUT = '\x1b]133;C\x07'`**

`command_status` is the command status, 0-255.

`shell_integration.`**`AFTER_OUTPUT = '\x1b]133;D;{command_status}\x07'`**

iTerm2 specific sequences. All optional.

`shell_integration.`**`SET_USER_VAR = '\x1b]1337;SetUserVar={user_var_key}={user_var_value}\x07'`**

The current shell integration version is 1. We don't use this as an outdated shell integration version would only prompt the user to upgrade the integration that comes with iTerm2.

`shell_integration.`**`SHELL_INTEGRATION_VERSION = '\x1b]1337;ShellIntegrationVersion={shell_integration_version}\`**

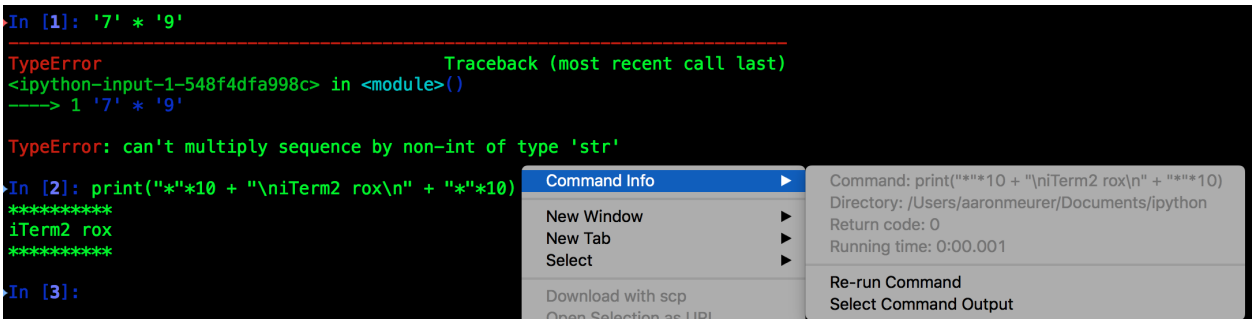
`REMOTE_HOST` and `CURRENT_DIR` are best echoed right after `AFTER_OUTPUT`.

`remote_host_hostname` should be the fully qualified hostname. Integrations should allow users to set `remote_host_hostname` in case DNS is slow.

`shell_integration.`**`REMOTE_HOST = '\x1b]1337;RemoteHost={remote_host_username}@{remote_host_hostname}\x07'`**

`shell_integration.`**`CURRENT_DIR = '\x1b]1337;CurrentDir={current_dir}\x07'`**

2.3 IPython



IPython shell integration extension

Enables iTerm2 shell integration in the IPython shell.

Note: This does not yet work with IPython 5.0. See https://github.com/asmeurer/iterm2-tools/pull/6.

To load, use:

```
%load_ext iterm2_tools.ipython
```

To load every time IPython starts, add:

```
try:
    import iterm2_tools.ipython
    c.TerminalIPythonApp.extensions.append('iterm2_tools.ipython')
except ImportError:
    pass
```

to your IPython configuration file.

Some notes about this:

- iTerm2's shell integration only supports single line commands. For multiline code, the first line will be saved as the command.
- The "Out" prompt will be included in the captured output. This is because the captured output is begun as soon as the code is executed. This is done so that text printed to stdout will be included (e.g., if you run "print('hello')" there will be no "Out" prompt).

- If an exception is raised, the command status will be set to 1 (making the iTerm2 shell integration arrow turn red). Otherwise it will be set to 0.
- However, due to a bug in IPython, SyntaxErrors will not register as failures (the arrow next to the prompt won't turn red).
- This requires a version of IPython greater than 4.0.0. Otherwise, due to a bug in IPython, the invisible codes printed in the prompt will be read by IPython as not invisible, causing the "Out" prompt to indent several characters (however, aside from this bug, it should work fine).
- This code adds a `set_custom_exc` handler to IPython to check the command status. IPython currently only supports one exc_handler at a time, so this may break other code that also uses this functionality.

2.4 Projects using iTerm2 tools

Some example projects using iterm2-tools. If you know of another project using this, please let me know.

2.4.1 catimg

Display an image of a cat from Imgur in your terminal. https://github.com/asmeurer/catimg

Install with

```
pip install catimg
```

2.4.2 giphycat

Display a gif from Giphy in your terminal.

Install with

```
pip install giphycat
```

2.5 Indices and tables

- genindex
- modindex
- search

2.5.1 License

MIT

i

A

B

C

D

I

O

P

R

S