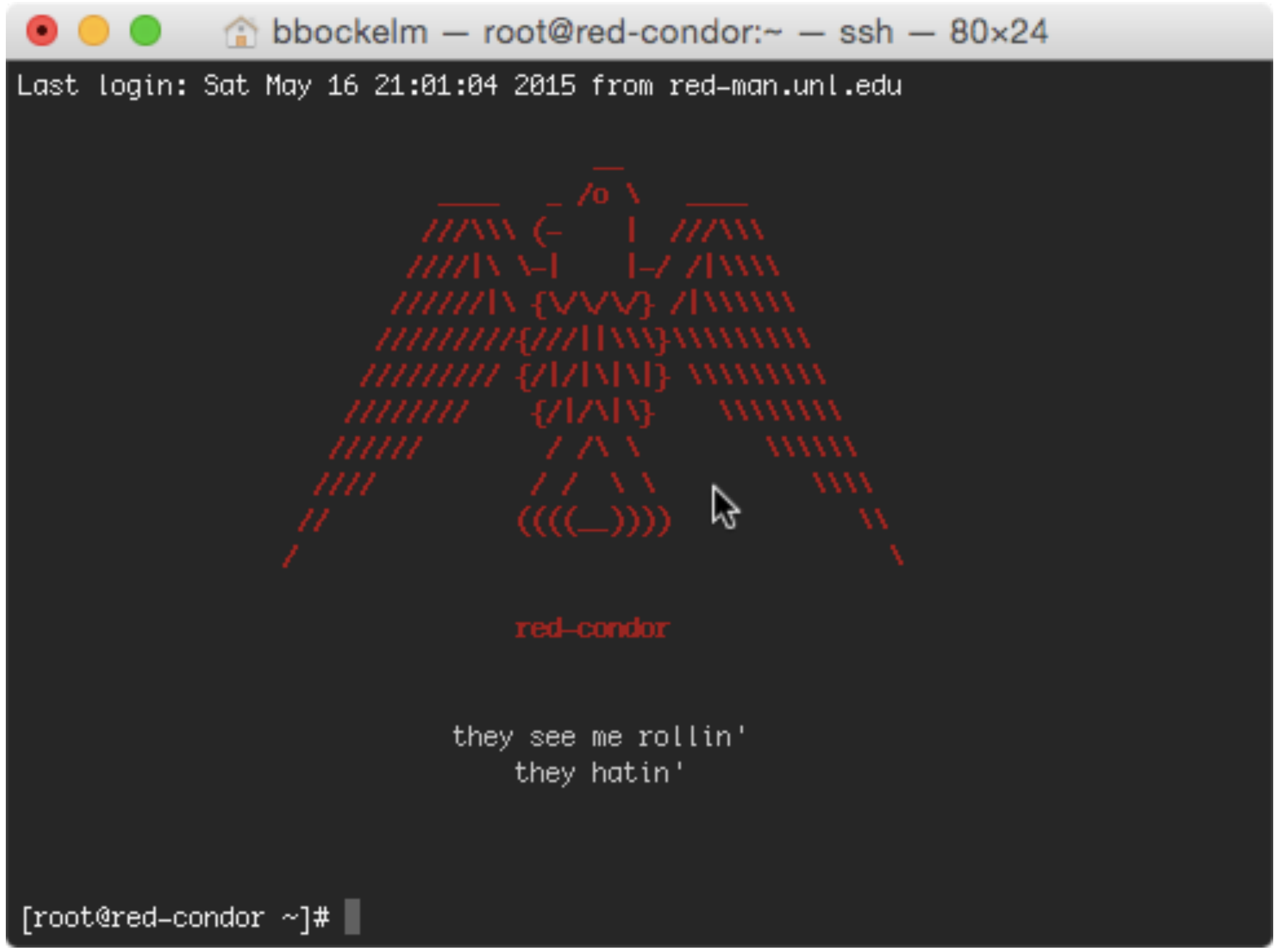
The History And Future of the Python Bindings

Brian Bockelman
HTCondor Week 2015



```
Last login: Sat May 16 21:01:04 2015 from red-man.unl.edu

                                      __
                           ____    _ /o \    ____
                          ///\\\ (-     |  ///\\\
                         ////|\ \-|     |-/ /|\\\\
                        //////|\ {VVVV} /|\\\\\\
                       /////////{///||\\\}\\\\\\\\\
                      ///////// {/|/|\|\|} \\\\\\\\\
                     ////////    {/|/\|\}    \\\\\\\\
                    //////        / /\ \        \\\\\\
                   ////          / /  \ \          \\\\
                  //            ((((__))))            \\
                 /                                      \


                                red-condor


                          they see me rollin'
                              they hatin'



[root@red-condor ~]#
```

WARNING

This is not a tutorial. This is a tutorial: http://research.cs.wisc.edu/htcondor/HTCondorWeek2013/presentations/Bockelman_Python.pdf

http://research.cs.wisc.edu/htcondor/HTCondorWeek2014/presentations/TheisenT-Python.pdf

This is an attempt to explain what the heck the python bindings are!

HTCondor Clients in 2012

Command Line Clients

Fully Featured!

Requires fork/exec and process handling

Outputs in multiple formats

Something Missing In The Middle

SOAP Clients

Features! (Some)

Language agnostic (everyone hates XML equally?)

Caveats with respect to scalability, security.

“Someday I want to play with boost.python. I heard it was really nice”

- Dan Bradley (paraphrased)
  July 12, 2012

Why Python?

- Plausible to do “on the side”: Clear, straightforward bridge to C++
    - HTCondor doesn’t have a “library”, so SWIG isn’t useful.
    - All could be done in C++; no python in python bindings.
- Anecdotally, one of the most popular sysadmin and integrator scripting languages.
- … because I wanted to!

Start reading up on
boost.python around Christmas

```
commit 5b95904e4963735628bbdd681d96b9f639c4f535
Author: Brian Bockelman <bbockelm@cse.unl.edu>
Date:   Wed Dec 26 17:02:17 2012 -0600

    First version of the ClassAd wrapper library.
```

31 commits later

```
commit a3e0bb1e7bebe16aef0c94ba966789843b330a11
Author: Brian Bockelman <bbockelm@cse.unl.edu>
Date:   Fri Jan 4 21:35:49 2013 -0600

    Add tests for old-ad parsing/printing and exprtree lo
```

One Week Later

- On January 3, moved from github into condor_contrib.
- Released with 7.9.4 on February 20, 2013.
- April 22, moved from condor_contrib to HTCondor core.

Design Philosophy

- **ClassAds**: Everything based on ClassAds; make these the “core” of the bindings.
- **pythonic**: Semantics and APIs should feel natural to a python programmer.
  - Use iterators, exceptions, guards. ClassAds behave as much like a dict as reasonable.
- **Backward compatible**: APIs are here to stay for as long as possible.
  - When we absolutely must, use standard python DeprecationWarning techniques.
- **Native code**: Call same HTCondor library code as CLI; identical in performance.
- **Complete**: If you can do it with the command line tools, you should be able to do it with python.

Base Features

- The first year of python bindings focused on:
  - **Representing ClassAds**: ClassAd and expressions as a python object; lazy-evaluation of expressions; natural conversions between equivalent ClassAd and python types.
  - **Scheduler Interaction**: Nuts-n-bolts - querying jobs, submitting jobs, hold/remove.
  - **Collector Interaction**: Query for ads, locate daemon(s).
- Already quite powerful! The most commonly used functions are likely from the first year of work.

Advanced Features

- More **complete ClassAd** bindings: esoteric topics like composing expressions from python, symmetricMatch, flatten, internalRefs.
    - Implement missing dictionary-like methods (e.g., update, setdefault, items).
    - Most other methods silently convert dictionaries to ClassAds.
    - Python functions can be registered for use by HTCondor daemon expressions.
- **Negotiator**: Retrieve & change priorities. Get / update resource usage.
- **Schedd**: history queries, refreshGSIProxy, transactions
- **Generic Daemons**: Remote daemon parameter get/set. Directly query daemon for ads.

Don’t memorize this slide - simply evidence that we take our “complete” goal seriously!

Schedd Transaction API

- One *powerful* concept in HTCondor is the two-phase commit. Provides consistency: prevents jobs from being in an unknown “half-submitted” state.
  - The power of this isn’t well-exposed from the CLI tools, but one secret power of HTCondor.
  - *But* transactions are now a first-class citizen!

```
with schedd.transaction() as txn:
  schedd.submit(ad, 5)
  schedd.edit(“1234.5”, “foo”, 4)
```

  - Everything within a transaction succeeds or fails.
  - Changes can be made durable (fsync) or non-durable.

Logs, Oh My!

- If you want job status without putting load on the schedd, you need to parse/follow the logs. These are a *complex* format; luckily, HTCondor has a handy reader implementation:
    - **LogReader**: Parse event objects (ClassAds) from a user/job log file
    - **EventReader**: Parse event objects from the schedd-wide log. Low-level & requires privileged access - but extremely powerful. Useful if you want to mirror/monitor entire schedd.
- You can use these in blocking (wait for events) or non-blocking mode. Can be used in event loops.

Improved Daemon Integration

- Python bindings started as targeting the “one-off” scripts that admins commonly write.
- However, they are starting to pop up more frequently in daemons. It’s often useful to make these user-written daemons behave more like a HTCondor daemon.
- We’re starting to see a bit of this already:
    - **Locks**: Implements the same locking logic in use by HTCondor. Useful if you need a write-lock on a log file to prevent the schedd from appending to it. CMS uses this to safely edit DAGMan log files.
    - **dprintf** from python: Access to the HTCondor logging subsystem.
    - **send_alive**: implements the child heartbeat protocol for `condor_master`.
    - **set_subsystem**: Act like a specific subsystem with respect to configuration or logging policies. Useful if you want to see schedd-specific configs.
- This is enough to write a simple HTCondor-like daemon that runs under the `condor_master`. However, this is just a preview of the line of thought…

Things That Happen With Age

- The python bindings are now old enough we have started to recognize design flaws. Example: **classad.parse()** method.
    - This is supposed to parse a single ClassAd from text input.
- What happens if you feed the method multiple ClassAds?
    - **Currently**: if parsing “new-style ads”, **ignore** “rest” of text.
    - If parsing “old-style ads”, throw **exception** on “rest” of text.

Things That Happen With Age

- Proposal: Sane, consistent parsing APIs:
  - **parseAds (existing)**: Input text or file, return an iterator of ads.
  - **parseNext** (8.5.x**?**): Input file-like object, return the next ad, advancing the file iterator.
  - **parseOne** (8.5.x**?**): Input text, return a single ad - merging all ads in input together.
- This will be our *first big test* of a function deprecation. Our goal is to have uses of **parse** raise a DeprecationWarning.
  - *Note* - python2.7 suppresses these warnings by default. We will override this depending on the HTCondor config knob `ENABLE_DEPRECATION_WARNINGS`.
  - Food for though: If this scheme is introduced in 8.5, should we remove 'parse' in 8.6.0 or 8.8.0?

Here Lie The Dragons

Future - Submit Language

- The first time a user toys with Schedd.submit, they learn of the *wide gulf* between the `condor_submit` language and job ClassAds (python uses the latter).
  - Many simple things in the submit language are awkward in the ClassAd.
  - `condor_submit` is surprisingly heavy-weight, auto-filling many attributes for you. Python users are on their own!
- **Goal**: Get condor_submit to behave like a library so I can invoke it directly from python. Both languages can be used by the python bindings.
  - The submit language is not very natural inside python (like ClassAds is). A pythonic API is going to be tough!

Future - CEDAR?

- HTCondor daemons all speak the same binary transport and application protocol, CEDAR. Provides basis for HTCondor security and message-passing semantics.
- I would like to provide python bindings for CEDAR.
- Why? CEDAR is the ‘assembly language’ for inter-daemon communication.
    - Prototype even wilder ideas such as making the match a first-class citizen in python.
    - Can “fuzz” daemons - possibly useful for testing?
    - Brian’s longtime dream: I can write a script that manually plucks an unclaimed startd from the collector and hand it to a schedd.

Future - DaemonCore?

- All common HTCondor daemons are written to use the same core event loop - “DaemonCore”.
  - One of about a dozen event loop implementations, but this provides the lovable behaviors and quirks of HTCondor.
- Python is starting to standardize event loop APIs.
  - Could we expose DaemonCore as one of these?
- Co-routines allow you to write code that looks blocking but behaves non-blocking.
- For inspiration, see: https://docs.python.org/3/library/asyncio.html
  - Will require python 3.4, so this is an “aspirational” and long-term.

Where can you help?

- The only thing better than feedback are patches!
- Places I'd love help:
  - (Better) python3 support.
  - Add more unit tests.
  - Get unit tests run inside the HTCondor tests.
  - Better/more examples in the documentation.