Writing a *fast* command line tool in Python

Andrew Bennetts

5th February, 2009
The Problem

What makes Python programs slow?

Ideas for improving Python itself

Summary

What do I mean by fast?
What do I mean by fast?

For Bazaar, fast means fast enough to be pleasant to use.
What do I mean by fast?

For Bazaar, *fast* means *fast enough to be pleasant to use*.

For this talk, I care about startup time.
Why startup time?

Bazaar is a command line program. It is used *frequently* and *interactively*.

So startup time hits users every time they invoke `bzr`. 
Digression: why not startup time

Obviously, shaving 100ms off startup time doesn’t make much difference if your program takes 20min to run.
How fast can Python be?
How fast can Python be?

Let’s time fastest possible python program: a program that does nothing.
How fast can Python be?

Let’s time fastest possible python program: a program that does nothing.

We can do better. What’s faster than nothing?
What makes Python programs slow?
(and what to do about it)
Imports

Imports are slow. How slow?
Why? Importing a module has to:
Why? Importing a module has to:

- find the module on disk
Imports

Why? Importing a module has to:

- find the module on disk
- load the bytecode
Why? Importing a module has to:

- find the module on disk
- load the bytecode
- *run* the module
Why? Importing a module has to:

- find the module on disk
- load the bytecode
- *run* the module
- and maybe even parse and compile the module first
Making imports faster

There’s lots of things you can do to make imports faster.
Making imports faster

First rule of optimisation:
Making imports faster

First rule of optimisation: profile
Making imports faster

First rule of optimisation: **profile**

bzr --profile-imports is my friend
Making imports faster

Ok, now we know which imports we’re doing and how slow they are.

Now what?
Making imports faster

Ok, now we know which imports we’re doing and how slow they are.

Now what?

In Bazaar’s case, the answer is: do fewer imports
Making imports faster: Lazy imports

from bzrlib.lazy_import import lazy_import
lazy_import(globals(), ""
import cStringIO
from bzrlib import import branch
..."""
Making imports faster: Lazy imports

```python
from bzrlib.lazy_import import lazy_import
lazy_import(globals(), ""
import cStringIO
from bzrlib import branch
...
"")
```
Making imports faster: Minimise dependencies

One way to do fewer imports is...
Making imports faster: Minimise dependencies

One way to do fewer imports is... to have fewer things to import.
Making imports faster: Minimise dependencies

One way to do fewer imports is... to have fewer things to import.

- Think about which third-party modules you depend on. Do you really need them?
Making imports faster: Minimise dependencies

One way to do fewer imports is... to have fewer things to import.

➤ Think about which third-party modules you depend on. Do you really need them?

➤ Find and kill unused import statements. **pyflakes** is great for this.
Quick Quiz: You’re writing a new package, mypackage, with lots of cool submodules.

Do you:
Making imports faster: Keep `__init__.py` empty

**Quick Quiz:** You’re writing a new package, `mypackage`, with lots of cool submodules.

Do you:

(a) Have an empty `mypackage/__init__.py` file.

Or...
Making imports faster: Keep `__init__.py` empty

(b) Promote everything possible into the `__init__` module.
Making imports faster: Keep `__init__.py` empty

(b) Promote everything possible into the `__init__` module.

```python
from mypackage.moduleA import Amber, Axolotl
```

Andrew Bennetts

Writing a fast command line tool in Python
Making imports faster: Keep `__init__.py` empty

(b) Promote everything possible into the `__init__` module.

```python
from mypackage.moduleA import Amber, Axolotl
from mypackage.moduleB import Berry, Brute
```
Making imports faster: Keep `__init__.py` empty

(b) Promote everything possible into the `__init__` module.

```python
from mypackage.moduleA import Amber, Axolotl
from mypackage.moduleB import Berry, Brute
... etc ...
```
Making imports faster: Keep `__init__.py` empty

(b) Promote everything possible into the `__init__` module.

```python
from mypackage.moduleA import Amber, Axolotl
from mypackage.moduleB import Berry, Brute
... etc ...
from mypackage.moduleZ import ZOMG, KitchenSink
```
Making imports faster: Do less work at import time

Importing a module *executes* that module. So:
Making imports faster: Do less work at import time

Importing a module *executes* that module. So:

Avoid expensive module-global code. A common culprit is pre-calculating expensive values.
Making imports faster: Do less work at import time

Importing a module *executes* that module. So:

Avoid expensive module-global code. A common culprit is pre-calculating expensive values.

Be wary of having lots and lots of cheap code too — it adds up.
Making imports faster: Do less work at import time

Importing a module *executes* that module. So:

Avoid expensive module-global code. A common culprit is pre-calculating expensive values.

Be wary of having lots and lots of cheap code too — it adds up.
The Standard Library — re.compile

re.compile pre-calculates an expensive value. Don’t do it at import time.

bzrlib.lazy.regex monkey-patches re.compile to delay the compile until the first use.
The Standard Library — slow imports

You might be surprised at how much time it takes to import parts of the standard library:

- **string**: 5ms to load what should be a trivial module
The Standard Library — slow imports

You might be surprised at how much time it takes to import parts of the standard library:

- **string**: 5ms to load what should be a trivial module
- **urllib**: 8ms. bzr often only needs `urllib.escape`, but `urllib` loads `socket` loads `ssl`.
The Standard Library — slow imports

You might be surprised at how much time it takes to import parts of the standard library:

- **string**: 5ms to load what should be a trivial module
- **urllib**: 8ms. bzr often only needs urllib.escape, but urllib loads socket loads ssl.
- **copy**: 30ms. Fixed in Python 2.5. In 2.4 it imports inspect, which imports tokenize, which uses most of the 30ms.
You might be surprised at how much time it takes to import parts of the standard library:

- **string**: 5ms to load what should be a trivial module
- **urllib**: 8ms. bzr often only needs urllib.escape, but urllib loads socket loads ssl.
- **copy**: 30ms. Fixed in Python 2.5. In 2.4 it imports inspect, which imports tokenize, which uses most of the 30ms.
- **distutils.sysconfig**: 2.6ms
The Standard Library — slow imports

You might be surprised at how much time it takes to import parts of the standard library:

- **string**: 5ms to load what should be a trivial module
- **urllib**: 8ms. bzr often only needs urllib.escape, but urllib loads socket loads _ssl.
- **copy**: 30ms. Fixed in Python 2.5. In 2.4 it imports inspect, which imports tokenize, which uses most of the 30ms.
- **distutils.sysconfig**: 2.6ms
- **locale**: 2ms
You might be surprised at how much time it takes to import parts of the standard library:

- **string**: 5ms to load what should be a trivial module
- **urllib**: 8ms. bzr often only needs urllib.escape, but urllib loads socket loads ssl.
- **copy**: 30ms. Fixed in Python 2.5. In 2.4 it imports inspect, which imports tokenize, which uses most of the 30ms.
- **distutils.sysconfig**: 2.6ms
- **locale**: 2ms
- **logging**: 3ms
You might be surprised at how much time it takes to import parts of the standard library:

- **string**: 5ms to load what should be a trivial module
- **urllib**: 8ms. bzr often only needs urllib.escape, but urllib loads socket loads _ssl.
- **copy**: 30ms. Fixed in Python 2.5. In 2.4 it imports inspect, which imports tokenize, which uses most of the 30ms.
- **distutils.sysconfig**: 2.6ms
- **locale**: 2ms
- **logging**: 3ms
- **xml.etree.cElementTree**: 5ms.
Ideas for improving Python itself
Add lazy_import to Python

Python really needs to include lazy_import (or something like it).
Add lazy_import to Python

Python really needs to include lazy_import (or something like it).

For example, Bazaar uses the ConfigObj library. ConfigObj unconditionally imports the compiler module for its “unrepr” feature.

Importing compiler wastes 10ms for a feature Bazaar never uses.

If lazy_import were built-in, ConfigObj could use it. Problem solved.
Add lazy_regex to Python

Lazy re.compile would be a good default.
Add bzrlib.profile_imports to stdlib

bzr --profile-imports is useful.

It’d be even more useful if it projects other than bzr could use it!

The code is fairly simple. See bzrlib.profile_imports
Cache module locations between runs

Every time Python runs, it has to rediscover where its modules live.

These locations almost never change. So why should Python redo the same work over and over?

C programs on Linux have solved this: /etc/ld.so.cache. It’s automatically kept up to date whenever I install/uninstall Ubuntu packages.
Ruthlessly examine Python startup, inc. site.py
Summary

1. Profile your imports
Summary

1. Profile your imports
2. Use lazy_import (or equivalent)
Summary

1. Profile your imports
2. Use lazy_import (or equivalent)
3. ...Profit?
Questions?