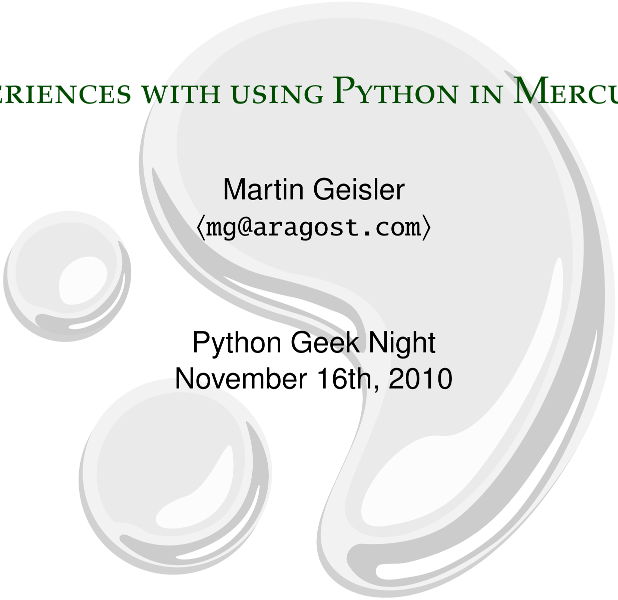
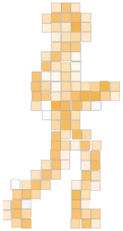
Experiences with using Python in Mercurial

Martin Geisler
⟨mg@aragost.com⟩

Python Geek Night
November 16th, 2010

About the Speaker

Martin Geisler:

- core Mercurial developer:
  - reviews patches from the community
  - helps users in our IRC channel

About the Speaker

Martin Geisler:

- core Mercurial developer:
  - reviews patches from the community
  - helps users in our IRC channel
- PhD in Computer Science from Aarhus University, DK
  - exchange student at ETH Zurich in 2005
  - visited IBM Zurich Research Lab in 2008

About the Speaker

Martin Geisler:

- core Mercurial developer:
  - reviews patches from the community
  - helps users in our IRC channel
- PhD in Computer Science from Aarhus University, DK
  - exchange student at ETH Zurich in 2005
  - visited IBM Zurich Research Lab in 2008
- now working at aragost Trifork, Zurich
  - offers professional Mercurial support
  - customization, migration, training
  - advice on best practices

Outline

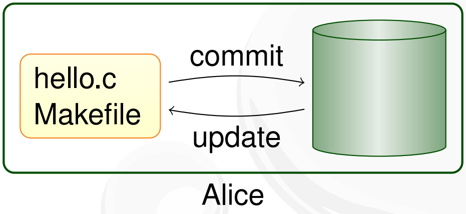
Outline

Mercurial in 3 Minutes

Mercurial is a distributed revision control system.

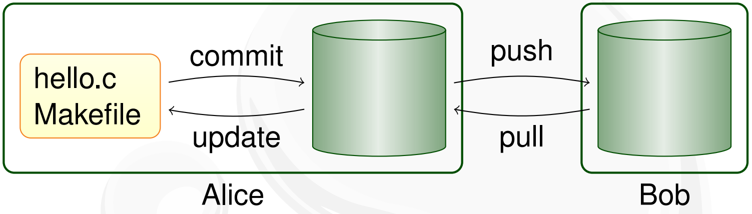
Mercurial in 3 Minutes

Mercurial is a distributed revision control system.



Mercurial in 3 Minutes

Mercurial is a distributed revision control system.



Who is Using it?

Mercurial is used by:

- Oracle for Java, OpenSolaris, NetBeans, OpenOffice, . . .
- Mozilla for Firefox, Thunderbird, . . .
- Google
- many more. . .

OpenOffice

Fairly large repository:

- 70,000 files
- 2 GB of data
- 270,000 changesets
- 2 GB of history

Mercurial is still fast on a repository of this size.

Outline

Rapid Prototyping

Python makes

- the revlog data structure in a 1 hour train ride

Cross-Platform Support

We want to support Windows, Mac, Linux, . . .

- Python's cross-platform support helped a lot

Clean Syntax

Python has a famously clean syntax:

- helps us write more clean code
- we have had contributors learn Python to write extensions
  - and they liked it!
  - result is lots of third-party extensions

Features per Line of Code

Python lets us get a lot done with little code:

- Mercurial:

Language	Lines	%
Python	62,205	95%
C	3,474	5%

- Git:

Language	Lines	%
C	151,354	95%
Shell	7,814	5%

Outline

Making Mercurial Start Fast

Starting Python:

```
$ time python -E -c 'print "."' > /dev/null
0.01s user 0.00s system 88% cpu 0.009 total
```

Making Mercurial Start Fast

Starting Python:

```
$ time python -E -c 'print "."' > /dev/null
0.01s user 0.00s system 88% cpu 0.009 total
```

Starting Mercurial with demandimport disabled:

```
$ time hg version -q > /dev/null
0.20s user 0.04s system 100% cpu 0.239 total
```

This delay is already very noticeable!

Making Mercurial Start Fast

Starting Python:

```
$ time python −E −c 'print "."' > /dev/null
0.01s user 0.00s system 88% cpu 0.009 total
```

Starting Mercurial with demandimport disabled:

```
$ time hg version −q > /dev/null
0.20s user 0.04s system 100% cpu 0.239 total
```

This delay is already very noticeable!
Starting Mercurial with demandimport enabled:

```
$ time hg version −q > /dev/null
0.04s user 0.01s system 100% cpu 0.048 total
```

Starting Mercurial

Even for printing the version string, Mercurial must do

- import its own modules
- load and parse $HOME/.hgrc
- import any extensions enabled by the user

Importing

Modules imported while starting

Python	17
Mercurial without demandimport	305
Mercurial with demandimport	69

I have enabled some typical extensions: bookmarks, churn, color, convert, gpg, graphlog, highlight, mq, patchbomb, progress, rebase, record, transplant

Demand-Loading Python Modules

Rewiring the import statement is quite easy!

```
import __builtin__
_origimport = __import__ # save for later

class _demandmod(object):
    """module demand−loader and proxy"""
    # ... one slide away

# modules that require immediate ImportErrors
ignore = ['_hashlib', '_xmlplus', 'fcntl', 'win32com.gen_py', ...]

def _demandimport(name, globals, locals, fromlist):
    """import name and return _demandmod proxy"""
    # ... two slides away

def enable():
    __builtin__.__import__ = _demandimport
```

Proxy Modules

```
class _demandmod(object):
    def __init__(self, name, globals, locals):
        object.__setattr__(self, "_data", (name, globals, locals))
        object.__setattr__(self, "_module", None)

    def _loadmodule(self):
        if not self._module:
            mod = _origimport(*self._data)
            object.__setattr__(self, "_module", mod)
        return self._module

    def __getattribute__(self, attr):
        if attr in ('_data', '_loadmodule', '_module'):
            return object.__getattribute__(self, attr)
        return getattr(self._loadmodule(), attr)

    def __setattr__(self, attr, val):
        setattr(self._loadmodule(), attr, val)
```

New Import Function

```
def _demandimport(name, globals, locals, fromlist):
    if name in ignore or fromlist == ('*',):
        # ignored module or "from a import *"
        return _origimport(name, globals, locals, fromlist)
    elif not fromlist:
        # "import a" or "import a as b"
        return _demandmod(name, globals, locals)
    else:
        # "from a import b, c"
        mod = _origimport(name, globals, locals)
        for x in fromlist:
            # set requested submodules for demand load
            if not hasattr(mod, x):
                submod = _demandmod(x, mod.__dict__, locals)
                setattr(mod, x, submod)
        return mod
```

Efficient Data Structures

Outline

Conclusion

Mercurial is a nice mix of Python and C code.