# 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

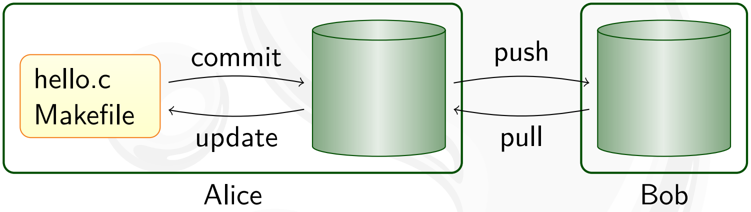
## Outline

# Outline

## Mercurial in 3 Minutes

Mercurial is a distributed revision control system:

- traditional systems (SVN, ClearCase, . . . ) have one server
- newer systems (Mercurial, Git, . . . ) have many servers



## Who is Using it?

Mercurial is used by:

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

## Who is Using it?

Mercurial is used by:

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

*Want to know more?*

Come to the free Mercurial Kick Start II!

| | |
|---|---|
| Date: | Wednesday, November 24th, |
| Place: | Technopark, Zurich |

See http://trifork.ch/

# Outline

## Advantages of Python

We like Python because of:

- rapid prototyping
  - the `revlog` data structure in a 1 hour train ride
- good cross-platform support
  - We want to support Windows, Mac, Linux, ...
- very clean syntax
  - easy to pick up for contributors

## Making Mercurial Start Fast

When you do `import foo`, Python does:

- search for `foo.py`, `foo.pyc`, and `foo.pyo`
- see if `foo.py` is newer than `foo.pyc` or `foo.pyo`
- load and execute found module
- do the whole thing recursively...

## Making Mercurial Start Fast

When you do `import foo`, Python does:

- search for `foo.py`, `foo.pyc`, and `foo.pyo`
- see if `foo.py` is newer than `foo.pyc` or `foo.pyo`
- load and execute found module
- do the whole thing recursively...

Starting Mercurial with `demandimport` disabled:

```
$ time hg version
0.20s user 0.04s system 100% cpu 0.239 total
```

This delay is already very noticeable!

## Making Mercurial Start Fast

When you do `import foo`, Python does:

- search for `foo.py`, `foo.pyc`, and `foo.pyo`
- see if `foo.py` is newer than `foo.pyc` or `foo.pyo`
- load and execute found module
- do the whole thing recursively...

Starting Mercurial with `demandimport` disabled:

```
$ time hg version
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
0.04s user 0.01s system 100% cpu 0.048 total
```

## Imported Modules

Effect of using `demandimport` on number of modules imported:

| System | Without | With |
|---|---:|---:|
| Python | 17 | — |
| Mercurial | 305 | 69 |

I have enabled 14 typical extensions where:

- `convert` pulls in Subversion and Bazaar modules
- `highlight` pulls in Pygments modules
- `patchbomb` pulls in email modules
- etc. . .

## Outline

## Optimizing Code

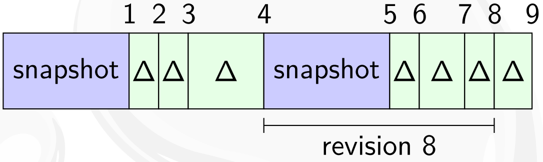
Start by profiling, then remove bottlenecks:

- use the right data structures
- add caches for data you reuse often
- rewrite in a faster language

## Efficient Data Structures

Mercurial avoids seeks since they are expensive:

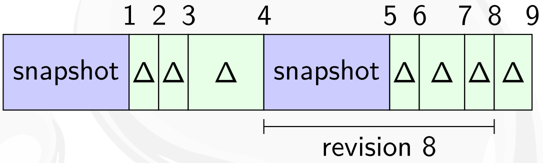
- any revision can be reconstructed with 1 seek and 1 read:



## Efficient Data Structures

Mercurial avoids seeks since they are expensive:

- any revision can be reconstructed with 1 seek and 1 read:



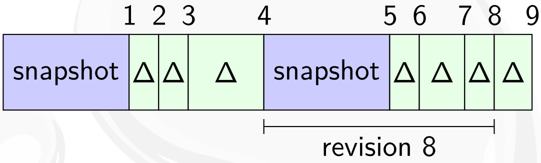
- directory order is maintained in repository:

| | | |
|---|---|---|
| `bar` | $\longrightarrow$ | `.hg/store/data/bar.i` |
| `baz` | $\longrightarrow$ | `.hg/store/data/baz.i` |
| `foo` | $\longrightarrow$ | `.hg/store/data/foo.i` |

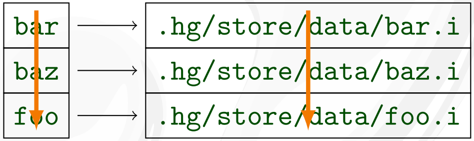
## Efficient Data Structures

Mercurial avoids seeks since they are expensive:

- any revision can be reconstructed with 1 seek and 1 read:



- directory order is maintained in repository:



optimizes use of kernel readahead

## Rewrite in Faster Language

If parts of your program are too slow, rewrite them!

$$\begin{tikzcd}
\text{Python} \arrow[dr] & & \\
& \text{C} \arrow[r] & \text{asm} \\
\text{Java} \arrow[ur] & &
\end{tikzcd}$$

Python embraces this hybrid approach:

- easy to build C extension modules with `distutils`
- Mercurial has six such extension modules

## Outline

## Conclusion

Mercurial is almost pure Python code:

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

Python makes it possible to strike a good balance between

- highly maintainable Python code
- performance critical C code

## Conclusion

Mercurial is almost pure Python code:

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

Python makes it possible to strike a good balance between

- highly maintainable Python code
- performance critical C code

**Thank you for the attention!**

## Conclusion

Mercurial is almost pure Python code:

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

Python makes it possible to strike a good balance between

- highly maintainable Python code
- performance critical C code

**Thank you for the atten**

Mercurial
Kick Start II
November 24th
trifork.ch

## OpenOffice

Fairly large repository:

- 70,000 files, 2,0 GB of data
- 270,000 changesets, 2,3 GB of history

Mercurial is still fast on a repository of this size:

```
$ time hg status
0.45s user 0.15s system 99% cpu 0.605 total
$ time hg tip
0.28s user 0.03s system 99% cpu 0.309 total
$ time hg log -r DEV300_m50
0.30s user 0.04s system 99% cpu 0.334 total
$ time hg diff
0.74s user 0.16s system 88% cpu 1.006 total
$ time hg commit -m 'Small change'
1.77s user 0.25s system 98% cpu 2.053 total
```

## 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', ...]

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, n, g, l):
        object.__setattr__(self, "_data", (n, g, l))
        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
```