☿ + 🐍 = ♥

Pourquoi python ?

Orignal Clipart: Dario Mitchell http://www.openbookproject.net/books/bpp4a

- Rapidité de développement
- Utile pour faire des PoC
- Multi plateforme
- Suffisamment rapide
- Itération rapides (refactoring)

markand yes it's sad that hg is in python

markand if it was in C, it would be the perfect scm :p

mpm If it was in C, it wouldn't exist because I would have spent 6 months getting a prototype working, rather than 2 weeks.

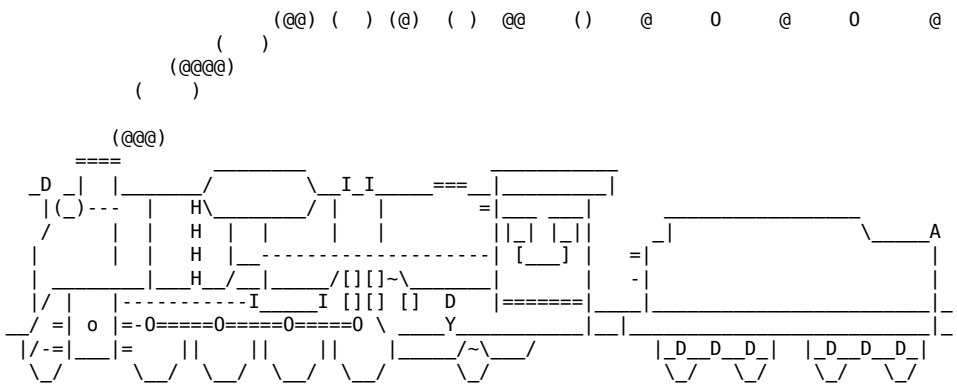
mpm (This comes from someone with over 20 years of C experience, including over 10 doing kernel development)

Timeline

- annonce du project: 19 avril 2005
- serveur http: 9 mai 2005
- windows: 9 juin 2005

Toi aussi, joue avec la CLI

- API officielle: la CLI
- Cohérente et stable.
- Comment faire des stats avec la CLI ?
- binding divers... vers la CLI



Quelques exemples

- Nombre de commits par auteur:

  $ hg log --template "{author}\n" | sort | uniq -c

  $ hg log --template "{author}\n" --rev "sort(all(), 'user')" | uniq -c

Revision sets

- Langage de requête pour sélectionner des révisions

```
hg log -r 'user(alexis@mozilla.com)'

hg log -r 'date(-7) and file("**/filename")'
```

- (et bien plus encore 🚀)

```
ancestors("2.8") - ancestors("2.7.3")

user(alexis@mozilla.com)
and FIREFOX_4_0b9_BUILD1::FIREFOX_4_0b9_RELEASE
and date("2011-01-08 to 2011-01-12")
and desc("fix")
and not merge()
```

Next level: python

Parce que des fois, c'est pas si simple.

```
# coding: utf8
import sys
from collections import defaultdict

from mercurial import hg
from mercurial.ui import ui


# On crée un objet dépôt, en utilisant le chemin passé en argument.
repo = hg.repository(ui(), sys.argv[1])

# … et on construit l'index des commits par personne.
stats = defaultdict(int)
for rev in repo:
    changeset = repo[rev]
    stats[changeset.user()] += 1

for user, count in sorted(stats.items()):
    print "{}\t {}".format(count, user)
```

L'extension facile

```
# coding: utf8
import sys
from collections import defaultdict
from mercurial import cmdutil

cmdtable = {}
command = cmdutil.command(cmdtable)

@command('stats', [], '')
def statscmd(ui, repo, **opts):
   '''Affiche le nombre de commit par contributeur.

   Trié du contribution décroissante.'''
    stats = defaultdict(int)
    for rev in repo:
        changeset = repo[rev]
        stats[changeset.user()] += 1

    for user, count in sorted(stats.items()):
        ui.write("{}\t {}\n".format(count, user))
```

Comparatif

- *hg log --template "{author}\n"* → **1,527 total**
- */usr/bin/python stats.py* → **0,762 total**

Le python ça se traine ?

Algorithmes qui poutre et langage de paille

- **add-remove → ×10**

  Ordre de traitement → ×2

  Raccourcis dans les comparaisons → ×5

- **Index sha1 → ×40**

  Structure de données dédiée

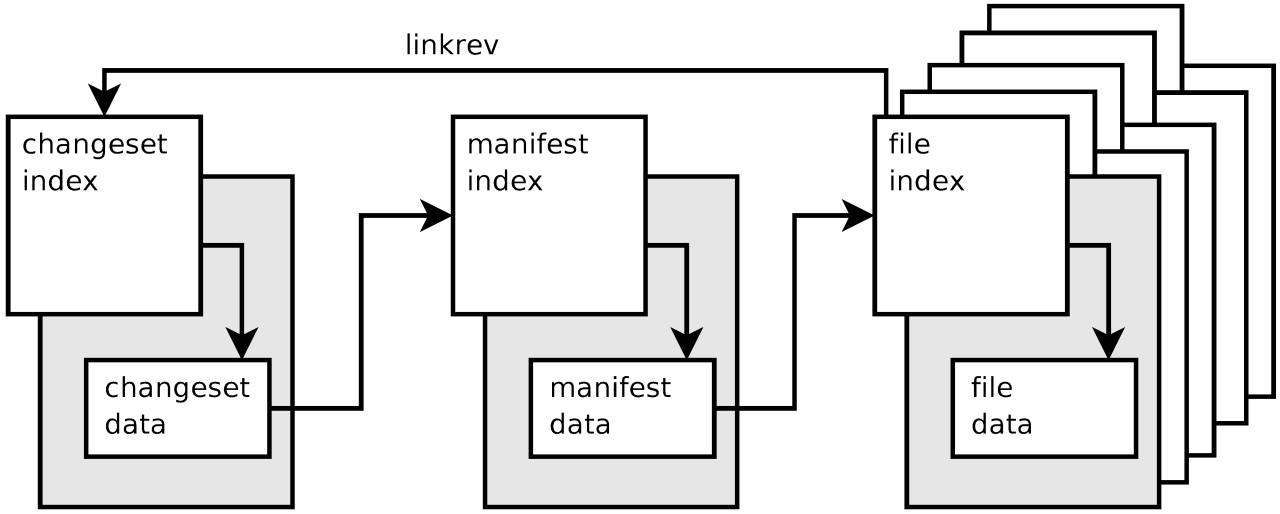
- **recherche de changeset caché → ×10**

  Code spécifique

Pas de bon algo sans bonnes structures de données

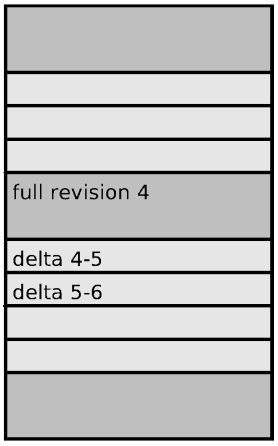
.

Hiérarchie des donnés



(Merci Monotone)

Stockage efficace



- **place:** O(total diff)
- **temps d'accès:** O(taille fichier)

- Append only
- Sens de parcours:

```
reversed(list(repo)): # →  1.34s
list(repo): # → 0.66s
```

Là où C bon

5% de C dans mercurial: (4K sloc [1])

- Lectures
- Écritures
- Différences
- Patches
- Indexes

Toujours sur des périmètres étroits et isolés

Version python via `--pure`

[1] pour 65K sloc python

Quelques problèmes liés à python

- Fonction
- Objet
- Parallélisme
- Allumage
- Import

Les appels de fonction, c'est long.

- 60 nano-secondes par appel
- 60 milli-secondes pour 1 million d'appels

hg log -l 100 sur mozilla central: 350 milli-seconde

La création d'objets, c'est la plaie.

```
for rev in cl:
    # changeset = repo[rev]
    # user = changeset.user()
    node = cl.node(rev)
    user = cl.read(node)[1]

    stats[user] += 1
```

8.328s → 5.324s (-47%)

Ton multicœur compte pour du beurre.

- Multithreading non parrallèle
- Processus: surcoût de communication et lancement.

Retard à l'allumage

- "Hello World" simpliste.

perl	3ms
ruby	4ms
python2.7	12ms
python3.3	23ms

Imports

- **hg --version** → 60ms
- **all import** → 130ms

setuptools et eggs ralongent sys.path 😿

Bazaar

- Le format m'a tuer [2]

[2](en partie) http://www.stationary-traveller.eu/pages/bzr-a-retrospective.html

The other DVCS ±±

- **moins rapide :** export, log, status,
- **équivalent :** taille, diff, rebase,
- **plus rapide :** clone, pull, update.
- **Parfois écrassant :** blame, 8s vs 50s [3] (format)
- **Bien plus rapide :** sur repo énorme **avec extensions** (de ×5 à ×10).

[3] En faveur de Mercurial, sur mozilla central ou le "mirroir" git à pourtant 5 fois moins d'historique

Python → Extensions "assez ouf"

Large file, gestion des gros fichiers binaires:

http://mercurial.selenic.com/wiki/LargefilesExtension

watchman, remplace toute la pile de status

https://bitbucket.org/facebook/hgwatchman

lz4log, ajoute une nouveau type de compression pour les dépôts

https://bitbucket.org/facebook/lz4revlog