Mercurial
Filosofía y visión de alto nivel

Jesús Cea Avión
Twitter: @jcea
jcea@jcea.es
http://www.jcea.es/

Python-Madrid, 20 de febrero de 2013

Antecedentes (I):

- ¿Qué es un sistema de control de versiones?

- VCS centralizados: CVS, SVN, etc.
  - Servidor centralizado, repositorio “canónico”.
  - Bloqueo de archivos.
  - Conflictos.
  - Grafo lineal. No hay “merge”.
  - No se puede trabajar desconectado.
  - Los no “core developers” tienen un “workflow” distinto.
  - Orientado a ficheros, no a “Changesets”
    - Se pueden crear combinaciones que nadie ha probado.
    - No es atómico.
    - “renames”.
    - La numeración de versiones es fichero a fichero.

Antecedentes (II):

- BitKeeper:
  - 2000: se anuncia, con licencia “curiosa”.
  - 2002: Migración de Kernel de Linux.
  - 2005: Cambio de licencia. Hay que pagar.

- Monotone, Arch, Darcs (haskell), Bazaar (fork de Arch).

- Mercurial, Git.

Factores diferenciadores (I):

- Un “workflow” recomendado.
  - Ventajas de una cultura uniforme entre proyectos.
  - Desarrollo orientado a “clones”. Eficiente en disco.
- Historia criptográficamente “segura”.
- Integración gráfica con MS Windows y MacOS X.
- Rápido. Criterio de diseño, formato de los deltas.
- ¡Programado en Python!.
  - Plugins.
- Zen de Python: difícil meter la pata, “una” forma correcta y obvia. Pero flexible y permite “abusos” premeditados.

Factores diferenciadores (II):

- No es necesario “housekeeping” ni conocer detalles de implementación.

- Compatibilidad del protocolo de red.

- Compatibilidad con el formato interno.

- Calendario de publicación de actualizaciones cumplido a rajatabla:
  - Una “major version” el día 1 cada tres meses.
  - Una versión “bugfix” el día 1 de cada mes. Como mínimo.
    - Actualizaciones fuera de ciclo si es necesario.
  - Evolución rápida.
  - Comunidad receptiva.

Factores diferenciadores (III):

- “Named Branches” eternos.
  - “bookmarks”.

- Intuitivo y bien documentado.

Factores diferenciadores (IV):

- Historia inmutable.
  - Grafo de historia complejo.
  - Historia divergente. ¿Cuál es el “head” correcto?
  - ¿Merge o Rebase?
  - Los cambios de historia locales son aceptables, los remotos no.
    - Fases: “public”, “draft”, “secret”.
    - Evolución de “changeset”: (experimental)
      - Uso seguro para novatos.
      - La idea es que la parte “mutable” de la historia se pueda compartir de forma simple y segura.
      - Marca “Changesets” como “obsoletos” y “unstable”.
      - “uncommit”, “fold”, “prune”, “touch”, “gdown”, “gup”, “evolve”.
    - Mercurial Queues. “Guards”. Versionado de parches.
    - “amend”.
    - histedit.
    - “hg convert”.

Ejemplos de “WorkFlows” (I):

- Ramas “estable” y “desarrollo”.

- “Cpython”
  - Repositorio canónico.
  - Clones.

- El problema de la historia divergente y múltiples “heads”.

- Ventajas de una historia lineal y “limpia”:
  - Fácil “review”.
  - “hg annotate”.
  - “hg bisect”.

Ejemplos de “WorkFlows” (II):

- “Cherry Picking”: “hg graft”.
- “tags”.
- Mercurial queues:
  - Se pueden versionar, pero vemos un “diff” de un “diff”.
  - “guards”.
  - Los parches se pueden reordenar.
  - Útil mover “changesets” entre la historia y la “cola”.
  - La resolución de conflictos no usa “three merge”.
  - Interacción con “fases”.
- “evolve” (I):
  - Experimental.
  - Historia parcialmente mutable y compartible.
  - Puede reemplazar, para mejor, “queues”. Salvo “guard”.

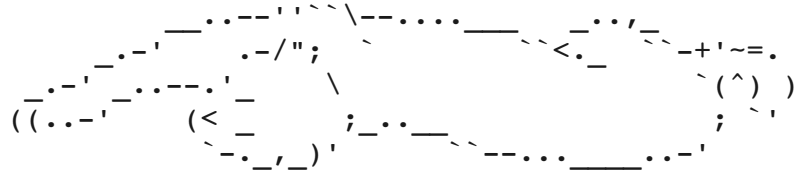
Ejemplos de “WorkFlows” (III):

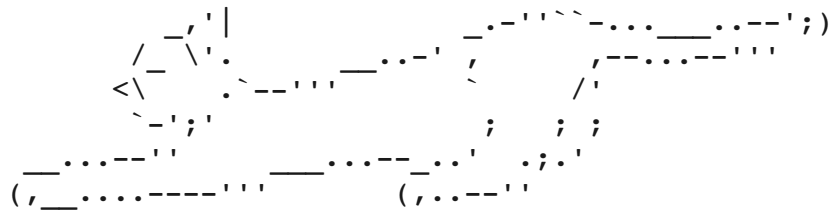
- “evolve” (II). Comparado con GIT:
  - Because you overwrite the git-branch, you have no conflict resolution. The last to act wins. This makes collaboration on multiple changesets difficult because you can’t merge concurrent updates on a changeset.
  - Every overwrite is a forced operation where the operator says, “yes I want this to replace that”. In highly distributed environments, a user may end up with conflicting references and no proper way to choose.
  - Because of this way to visualize a repository, git-branches are a core part of git, which makes the user interface more complicated and constrains moving through history.
  - Finally, even if all older changesets still exist in the repository, accesing them is still painful.

Conclusiones:

- El uso de un sistema de control de versiones es muy recomendable, incluso aunque seamos el único programador de un proyecto.
- Una forma “canónica” de hacer las cosas facilita la colaboración esporádica en proyectos. Otros proyectos siguen el mismo patrón (GitHub)
- La prioridad es la integridad de la historia y en ser “foolproof”. Pero puedes saltarte las restricciones.
- Evolución rápida, pero con una “visión”.
- Excelente auto-documentación.
- Vale la pena pensar con calma cómo queremos nuestro “workflow”, antes de empezar.
- ¡Está escrito en Python!.

Las fotos de gatitos de rigor...





¿Preguntas?

¿Ofertas de empleo? };-)