Sistemi di controllo versione per lo sviluppo software

Programmazione di Dispositivi Mobili a.a. 2012-2013

Claudio Pisa
claudio.pisa@uniroma2.it

Esempio: tesina

- File creati: tesina1, tesina2, tesina3
- Che succede se si lavora in più persone e ci sono due “tesina3”?
- E se il numero di file aumenta? Per esempio un progetto composto da 100 file?

Possibile soluzione

- Un'unica persona si occupa di mantenere una copia coerente del lavoro
  - Ogni persona che partecipa al progetto invia a questa persona la sua nuova versione del file

- Difficile capire cosa cambia tra una versione e l'altra dello stesso file
  - Idea: inviare invece che file completi, le differenze tra una versione e l'altra
  - Formato più diffuso per descrivere le differenze: **diff** o **patch**

Differenze: esempio

- listaspesa (revisione 1):
  - Pane
  - Latte
- listaspesa (revisione 2):
  - Uova
  - Latte
  - Zucchero
- listaspesa r2 – listaspesa r1:
  - Rimpiazzato pane con uova ed aggiunto zucchero dopo il latte

Diff e Patch: esempio

- diff -u listaspesa.original listaspesa

```
--- listaspesa.original 2011-12-13 12:41:17.315333710 +0100
+++ listaspesa   2011-12-13 12:42:06.142003082 +0100
@@ -1,2 +1,3 @@
-pane
+uova
 latte
+zucchero
```

- + e – ad inizio riga indicano rispettivamente le righe aggiunte e tolte
- diff -u listaspesa.original listaspesa > ingredienti_torta.patch

Diff e Patch: esempio

- cat listaspesa

```
pane
latte
```

- patch -p0 < ingredienti_torta.patch
- cat listaspesa

```
uova
latte
zucchero
```

Considerazioni

- Avere un'unica persona che si occupi di gestire I cambiamenti tra versioni è un'opzione, ma il processo si può (almeno parzialmente) automatizzare
- Come? Tramite dei software chiamati di “controllo versione” o “revision control” o “software di versioning”

revision control

- Alcuni tra I più usati:
  - CVS, subversion
  - mercurial, git, bazaar
- Usati da tutti i progetti software di medie e grandi dimensioni, sia open source che close source
- Esempi (open source):
  - Linux kernel → git
  - Android → git
  - Java (JDK) → mercurial
  - Ubuntu → bazaar
  - Mediawiki (il motore di wikipedia) → subversion

revision control

- Vantaggi:
  - Si può tornare indietro nel tempo
  - Con alcune tecniche si può capire in quale punto dello sviluppo di un software è stato introdotto un bug
  - Si può tenere traccia di chi ha fatto cosa
  - Si ha sempre almeno un backup del lavoro in corso

Subversion (svn)

- Initial release: 2000
- Modello client-server
  - Tutto il *repository,* ovvero la copia principale (e coerente) del progetto si trova su un server remoto
  - Ogni partecipante al progetto ha sul proprio PC una *working copy* sulla quale lavora
  - Alla fine di ogni fase di lavoro ogni partecipante invia i cambiamenti effettuati sulla propria working copy al server
  - Se ci sono *conflitti* tra versioni diverse dello stesso file, questi vanno risolti a mano
- Revisioni numeriche progressive

Subversion (svn)

- **checkout**: scarica una revisione (di solito l'ultima) dal server come working copy. Tipicamente viene fatto una sola volta all'inzio
- **update**: aggiorna la working copy dal server
- **diff**: mostra differenze tra due revisioni o le modifiche effettuate a partire dall'ultimo update
- **commit**: invia le modifiche al server

Subversion: workflow

$ svn checkout http...

$ <modifiche>

$ svn diff

$ svn update

$ <risoluzione di eventuali conflitti>

$ svn commit

git

- Inventato da Linus Torvalds et al.
- Initial release: 2005
- Modello distribuito
  - Ogni partecipante al progetto conserva sia una copia locale sia una copia di tutto il repository sul proprio PC
  - Non c'è bisogno di un server, ma ce ne possono essere più di uno
- Numeri di revisione forniti da una funzione di hash

Funzioni di hash

- y = H(x)
- Dove:
  - H funzione di hash
  - Output y ha una lunghezza fissa
  - Input x può avere una lunghezza qualunque
- Esempio: H = SHA1
  - H(“ciao”) → 1e4e888ac66f8dd41e00c5a7ac36a32a9950d271
  - H(“ciao mondo”) → 3249bfe01da50e9767daf9cd2a9685bed9756a53

git: comandi principali

- **init:** inizializza un nuovo respository git
- **clone**: copia un repository git esistente
- **remote add**: assegna un nome (es. “remote”) ad un repository remoto
- **checkout**: produce una working copy nella directory corrente
- **add**: aggiunge file allo *stage*
- **status**: mostra lo stato, ovvero file modificati, file in stage, etc
- **diff**: mostra differenze tra revisioni. Di default mostra il diff delle modifiche apportate ai file che sono in stage
- **commit**: inserisce nel repository le modifiche apportate ai file in stage
- **pull**: trasferisce commit da un repository git a quello corrente
- **push**: trasferisce commit dal repository corrente ad un altro repository git
- **log**: mostra la lista dei commit

git: workflow (semplice)

```
$ git clone http://
$ git checkout -b master
$ <modifiche>
$ git add <file modificato 1> <file modificato 2>
$ git diff
$ git commit
$ git push origin master
```

Git is more

- Tags
  - Etichette che si danno al codice sorgente del software, tipicamente usato per le release
- Branch and merge
  - Si possono sviluppare versioni parallele del software (branches) per tentare nuove strade, quali l'aggiunta di feature sperimentali
  - Se un branch funziona, allora i cambiamenti possono essere re-integrati (merge) nel ramo di sviluppo principale
- Firma digitale dei commit
- Integrazione con e-mail
- Git gui

References

- http://help.github.com/
- http://help.github.com/create-a-repo/
- http://git-scm.com/
- http://gitref.org/
- https://it.wikipedia.org/wiki/Controllo_versione
- https://github.com/pierpaolo-loreti/PDM2012