Pràctica 4: Control de versions i treball concurrent

Tecnologia de la Programació — iTIC

Aleix Llusà-Serra Sebastià Vila-Marta Marta Tarrés-Puertas

February 5, 2021

Contents

1 Organització

1.1 Objectius

El objectius d'aquesta pràctica són:

1. Entendre la importància de controlar el versionat durant la producció de programari.
2. Descobrir i analitzar la dificultat que comporta el desenvolupament de programari en equip pel que fa a la compartició del codi.
3. Aprendre a usar `subversion`, per solucionar els problemes anteriors.
4. Aprendre a usar una interfície, per exemple `trac` o `redmine`, com a complement de `subversion`.
5. Entendre que `subversion` pot ser usat també, de forma molt favorable, en altres circumstàncies com ara el manteniment de documents.
6. Entendre com `emacs`, l'editor, col·labora amb subversion i facilita l'ús d'aquest sistema.

1.2 Condicions

- La pràctica està calibrada per a ésser treballada en equips de dues persones.
- La durada de la pràctica és de 1 setmana.

1.3 Material necessari

Per dur a terme la pràctica cal tenir instal·lat, a banda del que hem usat fins ara, un client de subversion. En el cas del sistema operatiu GNU/Debian i d'altre que són equiparables cal que instal·leu el paquets anomenat `subversion`.

> TASCA PRÈVIA 1 Instal·leu en el vostre computador el paquet `subversion` usant l'ordre `apt-get` o `aptitude`. En els computadors de laboratori ja el trobareu instal·lat.

2 Introducció

Aquesta pràctica gira entorn de la gestió dels projectes de desenvolupament, un tema singularment important dins de l'àrea que es coneix com enginyeria del programari. En aquest apartat s'explica el context tecnològic en el que cal situar-se i alguns conceptes importants.

2.1 Cicle de vida i versions

Tots sabeu per l'experiència que teniu de les pràctiques anteriors que el programari es construeix de manera iterativa. Això és, es construeix una primera aproximació al producte que es vol i després, de forma iterativa es va polint i ampliant fins a obtenir el producte que es buscava. En aquest model de producció hi tenen un paper molt destacat els tests, que ens permeten anar comprovant la funcionalitat del programa minimitzant el risc de regressions.

També sabeu que, una vegada acabat el vostre producte, en condicions normals aquest entra en explotació, és a dir els clients comencen a usar-lo. Durant l'explotació el producte és susceptible

d'anar patint modificacions, ja sigui per esmenar errors que s'han detectat, per introduir millores o per adaptar-lo a noves circumstàncies.

Sigui com sigui, durant el *cicle de vida* d'un producte aquest no té sempre la mateixa forma sinó que evoluciona en el temps. Cadascuna de les formes per les que passa un programa durant la seva vida s'anomenen *versió*. [1]

Enregistrar l'evolució d'un producte i poder recuperar la forma que tenia en qualsevol moment de la seva vida és crucial per als programadors. És el que s'anomena *gestió de versions*. Fixeu-vos en el següent cas per entendre la seva importància:

> Un programa evoluciona en el temps seguint la següent línia de versions: A1, A2, A3, A4, A5 i A6, que és la darrera versió. A partir de la versió A3, el programa s'instał.la a casa dels clients i entra en explotació. Mentrestant, el programa es va millorant i per tant se'n van produint noves versions que succeeixen les anteriors. Les versions antigues no es conserven, ja que fer-ho comporta molts problemes organitzatius. En aquestes circumstàncies diferents clients tenen diferents versions. En particular, el client C1, té la versió A3 i el client C2 la versió A5.
>
> En un cert instant de temps, el client C2 en indica que ha detectat un error en l'aplicació. Aleshores sorgeixen els següents problemes:
>
> 1. Com recuperem la versió A5 per comprovar si l'error es produeix i perquè? Pot ser perfectament que en la versió A6, l'error no es manifesti.
> 2. Com sabem quines versions es veuen afectades pel mateix problema i, per tant, quins clients es veuen afectats?

Aquest cas il.lustra la necessitat de gestionar correctament l'evolució del programari. La importància és tal que existeixen perfils laborals especialitzats en aquest aspecte de la gestió dels projectes i també programari per ajudar a aquesta gestió. L'aplicació `Subversion`, per exemple, és una d'aquestes aplicacions anomenades *sistemes de control de versions*, que s'usen en aquest context.

2.2 Desenvolupament concurrent

Durant el desenvolupament d'un producte és molt habitual que hi participin diverses persones que, al mateix temps, van desenvolupant diferents parts del producte. És el que s'anomena *desenvolupament concurrent*. Aprendre a desenvolupar en aquest context i a treure'n el màxim profit és un dels secrets per produir a bon cost (i també per acabar les pràctiques a temps i amb un esforç raonable).

Aquest model de desenvolupament requereix que els diferents desenvolupadors puguin compartir fàcilment les dades dels seus projectes (codi font, documentació, etc.) de forma tal que quan un desenvolupador afegeix un fitxer o corregeix un error, la resta de desenvolupadors rebin les modificacions automàticament. Aquesta compartició de dades s'ha de poder fer, a més, sense que els desenvolupadors hagin d'estar en el mateix lloc de treball. Les eines informàtiques que faciliten aquesta compartició s'anomenen *eines de suport al desenvolupament cooperatiu*. Molt sovint aquestes eines van associades als gestors de versions.

[1] Els productes comercials també usen el concepte de versió i parlen, per exemple, de LibreOffice v2.4. El concepte de versió que usem aquí, però, fa referència a l'evolució durant la vida del programa. En certa manera, per passar d'una versió comercial a la següent han hagut de passar moltes versions de les nostres.

2.3 La gestió integral d'un projecte

Gestionar un projecte de desenvolupament de programari comporta multitud de tasques complexes. De la seva efectivitat en depèn en gran mesura l'èxit o fracas del projecte. A més de les tasques relacionades amb la gestió de versions o el desenvolupament concurrent es poden citar:

- La gestió de la documentació associada al projecte, tant tècnica com d'usuari.
- La gestió de la correctesa, mitjançant bateries de test i altres estratègies.
- La gestió de les tasques de desenvolupament, que s'encarrega de planificar i repartir les diferents tasques a les persones que formen l'equip de treball.
- La gestió de les incidències (errors, suggerències i altres informacions que provenen de l'usuari final. Sovint desencadenen depuració d'errors i millores.
- La gestió del cicle de release, que determina en quin moment caldrà llençar una nova versió als usuaris finals i quines novetats/correccions ha d'incorporar.

Per a la gestió integral d'un projecte s'usen conjunts d'eines específiques que integren moltes funcions útils en aquest context. Molt sovint aquestes funcions són accessibles a través d'una interfície web. Aquestes eines es coneixen com a *gestors de projectes*. `Redmine`, per exemple, és un gestor de projectes.

La gestió integral de projectes no és l'objectiu d'aquesta pràctica. Amb tot s'usarà parcialment un gestor de projectes per a algunes de les funcions.

3 Control de versions: el workflow habitual

La majoria de sistemes de control de versions, com el cas de `subversion`, són aplicacions client-servidor. El servidor és essencialment un dipòsit que emmagatzema fitxers i directoris així com la seva evolució en el temps. El client és l'aplicació que fa servir l'usuari per comunicar-se amb el servidor.

En aquesta arquitectura els programadors sempre mantenen una còpia local del projecte en el seu lloc de treball, còpia que se sincronitza amb el dipòsit usant l'aplicació client.

En aquest context hi ha molts patrons de treball possibles i se n'usen uns o altres segons les necessitats de gestió particulars del projecte. Això no obstant hi ha un workflow bàsic que es descriu a continuació.

El principi de funcionament és el següent:

1. El programador P, en el seu lloc de treball, crea els fitxers F1 i F2 en el directori D.
2. P declara que vol sotmetre al control del sistema de versions D, F1 i F2 (operació *add*). Sempre cal declarar explícitament que es desitja tenir un cert fitxer sota el sistema de control de versions. D'aquesta manera el desenvolupador tria quins són els fitxers importants dels que cal controlar-ne el versionat.
3. Quan P ho creu convenient, puja els fitxers al dipòsit (operació *commit*). És a dir, comunica al sistema de control de versions que vol enregistrar la versió dels fitxers tal i com estan en aquest instant.
4. P continua modificant els fitxers i, quan ho creu convenient, repeteix el punt 3 enregistrant una nova versió.

D'aquesta manera, el dipòsit va enregistrant els estats successius (versions) pels que van passant els fitxers en qüestió. Això permet després recuperar versions antigues, veure quins canvis s'han fet, etc.

Un segon programador Q pot demanar al dipòsit una còpia nova del projecte en el seu lloc de treball (operació *checkout*). Llavors, aquest pot treballar sobre aquesta còpia i, quan ho creu convenient, pujar al dipòsit les noves versions. Això permet a P reconciliar (operació *update*) la seva còpia del projecte i incorporar el treball de Q.

Durant aquesta pràctica s'exercitarà aquest workflow de treball.

4 Guió de treball

En aquesta pràctica intervenen diferents eines:

- `Redmine`, un sistema de gestió de projectes amb interfície web. Només se n'usarà una petita part, la que correspon al servei de control de versions. La instància de redmine que usareu es troba hostatjada al centre de càlcul de l'EPSEM sota la url `http://escriny3.epsem.upc.edu` i està dedicat específicament als projectes dels estudiants. Més endavant s'explicarà com usar-lo.

- `Subversion`, un sistema de control de versions client-servidor. Cada equip de treball disposarà del seu dipòsit de control de versions específic que mantindrà durant tot el quadrimestre. Com redmine s'entén bé amb subversion, usarem redmine per gestionar el dipòsit de versions de cada equip: crear-lo, consultar-lo, donar d'alta usuaris, etc.

La pràctica introdueix moltes eines i cal estar concentrar per entendre amb precisió què s'està fent i on en cada moment.

4.1 Alta dels usuaris i del projecte a redmine

El primer que cal fer és crear un grup de treball a redmine. Per fer-ho cal seguir els següents passos:

1. Donar-se d'alta com a usuari de redmine (tots els membres de l'equip).

2. Un membre, que esdevindrà el propietari del projecte, ha de crear un projecte a redmine.

3. El propietari ha d'afegir al projecte al resta de persones de l'equip i ha de definir el seu perfil.

Seguidament es detallen aquestes tasques.

Tasca 2 Doneu-vos d'alta com a usuaris de redmine. Cal que us connecteu a `http://escriny3.epsem.upc.edu` i que cliqueu damunt de Entra, just a la cantonada superior dreta de la pantalla. Doneu les credencials UPC, les mateixes que useu per a entrar a Atenea. Si tot va bé us demanarà algunes dades personals: nom, cognoms, email, etc. Contesteu-les.

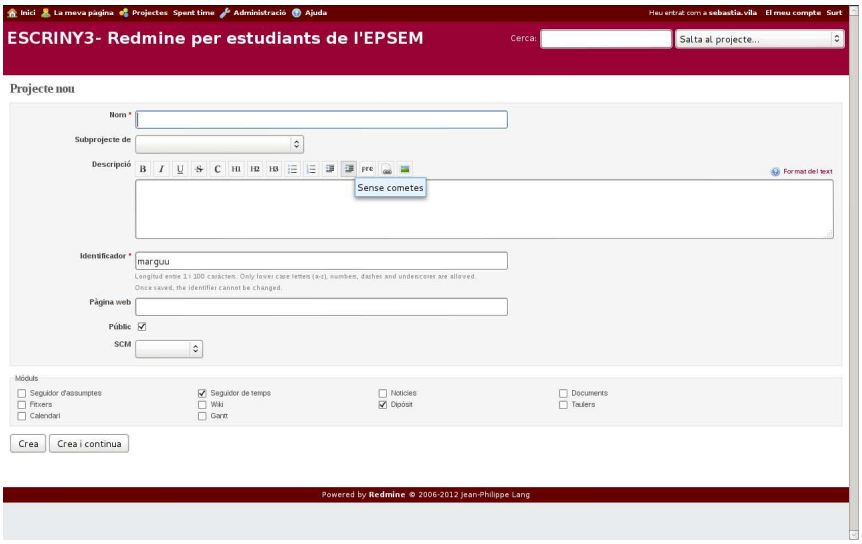

Figure 1: Formulari de creació de projectes a redmine

Tasca 3 Per crear un projecte, cal que algun membre del l'equip faci login a redmine i faci clic damunt de projecte (dalt a l'esquerra). Això el porta a el formulari de gestió de projectes. En aquest formulari, a la dreta hi ha un enllaç que indica Projecte nou. Cal fer clic damunt per crear un nou projecte. Això us porta al formulari de creació de projectes, que té l'aspecte que es mostra a la figura 1.

Per a donar d'alta el projecte cal que contesteu:

- El nom del projecte. Un text qualsevol que serà el nom públic del projecte. El podeu canviar més endavant.
- Una descripció del projecte. Un text breu que explica el projecte, qui el desenvolupa, i qualsevol altra cosa interessant que descrigui el projecte.
- L'identificador del projecte. *És un camp important.* Defineix un string que identifica el projecte de forma única dins de tots els projectes gestionats per redmine. Aquest identificador formarà part de totes les URL's del projecte i per tant convé que sigui fàcil de recordar. Si ja està ocupat no donarà d'alta el projecte. Una vegada fixat, l'identificador no es pot modificar.
- La propietat de public. Indica si el projecte és públic o privat. És a dir si altres persones fora del projecte poden consultar o no el projecte. Trieu allò que considereu escaient. Es pot canviar més endavant.
- El SCM[2]. *Camp molt important.* Determina si el projecte necessita d'un sistema de control de versions i, si és el cas, quin. En el vostre cas heu de seleccionar sempre *Subversion.* Això us garanteix que, després de crear-se el projecte també es crearà un dipòsit centralitzat de versions associat al projecte i gestionat amb subversion.

La resta de camps del formulari els heu de deixar com estan. Ara mateix no és necessari activar-ne cap altre. Una vegada emplenat el formulari creeu el projecte. Anireu a parar a la pàgina de gestió del projecte, que té un aspecte com el que s'indica a la figura 2.

[2]Acrònim de *Source control management*, una altra forma de dir sistema de control de versions.

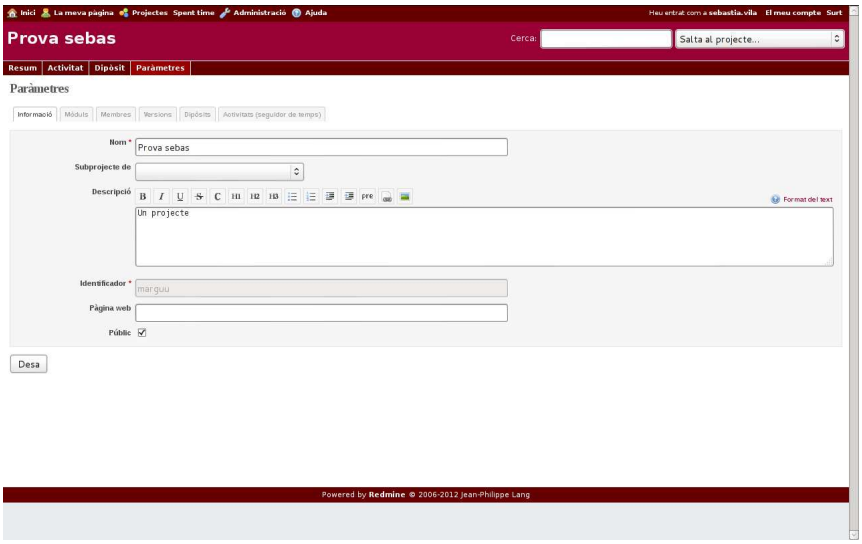

Figure 2: Pàgina de gestió d'un projecte redmine.

TASCA 4 Doneu d'alta la resta de persones de l'equip. A tal efecte cal que el propietari del projecte seleccioni el projecte, trii la pestanya de Paràmetres i, dins d'aquest formulari la pestanya de Membres. A la dreta us apareixerà la llista de membres del portal. Seleccioneu les persones que voleu incorporar al projecte i també quin rol voleu que tinguin dins del projecte. En el vostre cas el rol de Manager és l'adequat ja que dona pràcticament els mateixos permisos sobre el projecte que té el seu propietari. Ara les persones incorporades haurien de tenir accés al projecte.

Amb aquesta darrera tasca heu acabat la creació d'un projecte i el seu dipòsit de subversion a través de redmine. Fixeu-vos que l'estructura de redmine que hem usat es pot descriure amb un model UML com el de la figura 3.

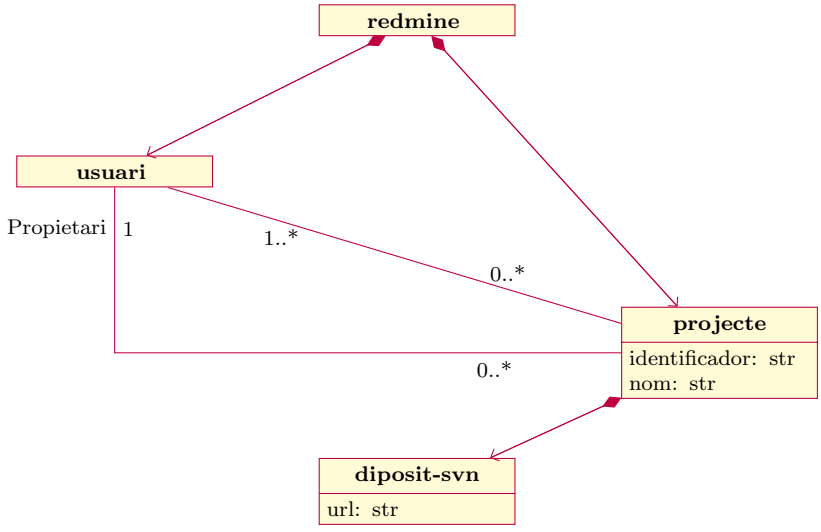

Figure 3: Model UML de redmine

Si tot el procés ha anat bé, a més d'un projecte a redmine heu de tenir un dipòsit de subversion associat al projecte. El dipòsit de subversion s'identifica mitjançant una url que està relacionada

amb la url del projecte. Per exemple, si l'identificador que heu definit pel vostre projecte és `idepro`, llavors la url del projecte a redmine és `http://escriny3.epsem.upc.edu/projects/idepro` i la url del dipòsit de subversion associat és `http://escriny3.epsem.upc.edu/svn/idepro`.

Si us connecteu a aquesta url us contestarà la interfície web del servidor de subversion. Ja notareu que és una interfície una mica gore que habitualment no usarem. Observeu que si el projecte és privat us demanarà que us identifiqueu: si és el cas heu d'usar les vostre credencials UPC, recordeu-ho!

4.2 Primeres passes amb subversion

Per començar a entendre com treballa subversion seguireu un guió que emula un desenvolupament de software. El guió és molt fàcil de seguir però *és molt important que el seguiu poc a poc amb plena consciència de què esteu fent i per què*. Altrament no us servirà de res i més endavant tindreu problemes amb l'ús del sistema de versions.

La feina encomanada serà la implementació d'una classe `Python` per representar fraccions així com d'un doctest en un fitxer separat. Això però, només és una bona excusa per experimentar amb el sistema de gestió de versions. Cal treballar des de la *terminal de GNU/Linux*.

Cada persona de l'equip ha de treballar independentment amb la seva identitat i en el seu computador. Aquí a l'enunciat anomenarem els membres de l'equip com U1 i U2 (usuari 1 i 2).

TASCA 5 Determineu qui fa el paper d'U1 i qui fa el d'U2. No us intercanvieu el paper durant tota la pràctica. Recordeu que cadascú ha de treballar al seu computador.

4.3 Creació de les còpies locals del dipòsit

TASCA 6 Cada usuari, U1 i U2, cal que es crei una còpia local del directori de treball de l'equip E. Per això un i altre usuari fan en els seus respectius computadors el següent (cada grup ha de substituir `<E>` pel nom de l'equip) després d'haver-se situat en el directori en el que volen treballar:

```
$ svn checkout --username 'ramon.llull' http://escriny3.epsem.upc.edu/svn/<E> <E>
```

Fixeu-vos en l'ordre:

svn és l'ordre que permet treballar amb el client de `subversion`. La part de subversion que s'executa en el vostre computador.

checkout és una subordre del client que significa fes-me una còpia local.

–username és una opció que permet indicar amb quin nom d'usuari cal identificar-se davant del servidor. A l'exemple s'usa el nom `ramon.llull`. En el vostre cas heu d'usar el vostre identificador UPC. Si el projecte és privar us demanarà la contrasenya. Si el projecte l'heu definit com a públic no serà necessari indicar el nom d'usuari.

Noteu també que l'ordre usa l'url `http://escriny3.epsem.upc.edu/svn/<E>` que és l'adreça del vostre servidor de subversion, l'encarregat de gestionar el dipòsit centralitzat de versions.

Després de fer aquesta comanda a cada usuari li ha d'haver aparegut un directori de nom `<E>`. Ara, fixeu aquest directori com a directori de treball fent:

```
$ cd <E>
```

Aquest nou directori és un directori *sota control* del sistema de versions.

4.4 Creació de l'estructura

Tasca 7 L'usuari U1, en el directori `<E>`, crea un subdirectori per emmagatzemar aquesta pràctica que es dirà `pract4`. Això es pot fer d'aquesta manera:

```
$ mkdir pract4
```

A continuació, cal enregistrar aquest nou directori a `subversion` ja que volem que sigui un directori sota control del sistema. Ho fem amb l'ordre:

```
$ svn add pract4
```

Noteu que no és necessari indicar de nou l'url del dipòsit, ni el nom d'usuari ni la contrasenya. El sistema local de subversion sap que esteu treballant en un directori sota control i recorda quin servidor el controla i com us identifiqueu en el servidor.

Fixem com a directori de treball `pract4` i, usant l'editor de textos, l'usuari U1 crea un fitxer que contindrà la classe `Python` que hem d'implementar. El fitxer s'ha de dir `fraccio.py`. En aquest fitxer només hi escriu la capçalera, una cosa com ara aquesta:

```
# -*- coding: utf-8 -*-
"""
=============
Mòdul fracció
=============
"""
```

Després de desar-lo cal tornar a la terminal, i enregistrar el nou fitxer que hem creat en el sistema de control de versions fent:

```
$ svn add fraccio.py
```

Fins ara totes les modificacions que hem fet són *locals*: s'han fet el el computador d'U1 però el servidor de versions no en té constància. L'ordre `add` no actualitza el dipòsit, simplement anota que volem mantenir el fitxer corresponent sota el control de versions. Els directoris i fitxers creats encara existeixen solament al computador d'U1. Imaginem que les donem per bones i les volem transmetre al servidor. Simplement cal que ens situem en el directori `<E>` i fem:

```
$ svn commit -m "Afegida l'estructura de la pràctica 4"
```

L'ordre `commit` cal entendre-la com *dóna per bo el que tinc, puja-ho al dipòsit i enregistra una nova versió*. L'opció `-m` és obligada i s'usa per documentar breument quins canvis hem fet des de l'anterior commit. Si no ho ha fet abans, per fer un commit us demanarà que us autentifiqueu. Feu-ho amb les credencials corresponents.

4.5 Redmine espia el dipòsit de subversion

El gestor de projectes redmine té constància de que el vostre projecte usa un sistema de control de versions i sap quin és el dipòsit que hi està associat. Com una eina de suport més a la gestió del projecte redmine visualitza via web l'històric del dipòsit de versions. Si ara us hi connecteu podreu veure la versió que acabeu de pujar.

Tasca 8 Connecteu-vos a redmine, trieu el vostre projecte i exploreu la pestanya dipòsit. Us mostrarà l'estat del vostre dipòsit i l'històric de versions. Sabreu qui ha fet quina feina a cada moment.

Si ara trieu la pestanya activitat podreu veure les diferents accions importants que afecten el projecte. En particular podreu veure la creació de noves versions fruit de les ordres de `commit`.

4.6 U2 s'actualitza

El computador d'U1 està sincronitzat amb el dipòsit. El que hi ha en el seu computador és el mateix que en la darrera versió del dipòsit. No passa el mateix amb U2. Ara U2, conscient de que no té la darrera versió en el seu computador vol actualitzar-la.

TASCA 9 Simplement cal que se situi en el directori `<E>` i ho demani a `subversion` fent:

```
$ svn update
```

Com per art de màgia, en el nostre directori apareixerà un subdirectori `pract4` i dins d'aquest un fitxer `fraccio.py`. Comproveu-ho.

4.7 U2 treballa

TASCA 10 U2 està animat i es posa a treballar. Se situa en el directori `pract4` del seu computador i amb l'editor obre el fitxer `fraccio.py` i hi afegeix l'esquelet de la classe Fraccio. Noteu com la capçalera ja hi és. L'havia escrit abans U1:

```
# -*- coding: utf-8 -*-
"""
=============
Mòdul fracció
=============
"""

class Fraccio(object):

    def __init__(self, n, d=1):
        self._num = n
        self._den = d
        self._simplifica()

    def _simplifica(self):
        pass

    def __add__(self,a):
        pass

    def __sub__(self,a):
        pass
```

Content per la feina feta, una vegada desat el fitxer, U2 vol enviar els canvis al servidor i ho fa així:

```
$ svn commit -m "Afegit l'esquelet de la classe Fraccio"
```

TASCA 11 Comproveu a través de redmine que els canvis han arribat al servidor del sistema de control de versions.

4.8 U1 se sincronitza

Tasca 12 U1 té ara la seva còpia local antiquada. Per actualitzar-la simplement executa:

```
$ svn update
```

Misteriosament en el fitxer `fraccio.py` de U1 apareix l'esquelet de classe que U2 havia escrit i pujat.

4.9 U1 i U2 treballen alhora

En aquest apartat explorem una de les funcions més importants del sistema de control de versions que facilitat enormement el treball concurrent. Llegiu primer l'apartat mirant d'entendre què es fa exactament. Després feu el que es diu en l'ordre que s'indica.

Tasca 13 Ara U1 es posa a desenvolupar part de la classe Fraccio en la seva còpia local. Exactament implementa el mètode _simplifica(). Mentrestant U2 també modifica la seva còpia local. Modifica el seu fitxer `fraccio.py` implementant el mètode __add__(). U1 i U2 treballen concurrentment, cadascú en la seva màquina, retocant cadascú la seva còpia local del mateix fitxer.

U2 afegeix a `fraccio.py` el següent:

```
def __add__(self, a):
    nd = self._den * a._den
    nn = self._num * a._den + a.num * self._den
    return Fraccio(nn, nd)._simplifica()
```

Mentrestant, U1 escriu a `fraccio.py`:

```
def _simplifica(self):
    a = self._num
    b = self._den
    while a != b:
        if a > b:
            a -= b
        else:
            b -= a
    return Fraccio(self._num/a, self._den/a)
```

U2, que acaba primer, puja la seva aportació al servidor fent:

```
$ svn commit -m "Implementat mètode add"
```

Per la seva banda U1, que acaba després, puja també la seva aportació fent:

```
$ svn commit -m "Implementat mètode _simplifica"
```

Ara però, el commit d'U1 és impossible i l'ordre diu quelcom semblant a:

```
Sending fraccio.py
svn: Commit failed (details follow):
svn: File 'fraccio.py' is out of date
```

Això ens diu que la versió que hem modificat és més antiga que la darrera versió que hi ha en el servidor. Recordeu que la versió que hi ha al servidor és la que acaba de pujar U2: comproveu-ho a través de redmine.

En aquestes condicions, U1 no pot pujar les modificacions. Primer cal que U1 actualitzi la seva versió local. Compte però! la versió versió local d'U1 conté modificacions, l'usuari U1 hi ha implementat el mètode _simplifica(). Què succeirà si actualitza la còpia d'U1? perdrem el treball?

Per comprovar-ho U1 executa l'ordre:

```
$ svn update
```

L'ordre funcionarà correctament i ens dirà quelcom similar a:

```
G fraccio.py
Updated to revision 5.
```

La `G` indica que hi ha hagut una fusió (*merge*). Mentre actualitza, fusiona els nostres canvis amb els que hi ha hagut al dipòsit. Si U1 mira el resultat fent:

```
$ less fraccio.py
```

veureu que el fitxer té ara el que U1 havia afegit i també el que U2 havia pujat al dipòsit. Ara tot es correcte i U1 pot pujar els seus canvis fent:

```
$ svn commit -m "Implementat mètode _simplifica"
```

Fixeu-vos que el procés de fusió automàtic permet anar incorporant els canvis que van fent la resta de persones de l'equip a la nostra còpia local sense perdre el que hem escrit nosaltres. És una eina molt important.

El procés de fusió funciona correctament en la majoria de casos. Això no obstant, hi ha vegades en que és necessària la intervenció manual. Entrar en aquest tema, però, duria la pràctica massa enllà.

5 Algunes consideracions

5.1 Quins fitxers han d'estar sota control?

Com heu vist, cal decidir explícitament quins fitxers i directoris han d'estar sota control del sistema de versions. Això és deliberat. La regla general diu que han d'estar sota control *només aquells fitxers del projecte que NO poden generar-se automàticament a partir d'altres*. És molt important per a la bona organització del projecte seguir aquest punt.

En particular, per exemple, cal afegir els fitxers `.py` però no els `.pyc`, que es generen automàticament a partir dels primers quan s'executa el programa. Tampoc s'han d'afegir els que acaben en titlla, que els genera automàticament `emacs` com a mesura de seguretat.

En el cas d'una documentació en restructured text, per exemple, afegiriem el fitxer `.rst` però mai el fitxer `.pdf` que hem obtingut processant el primer.

De la mateixa manera mai s'afegeixen fitxers que podem obtenir directament de la web com ara manuals, documents o enunciats de pràctiques.

Fixeu-vos que podeu tenir fitxers a la vostra còpia local sense que estiguin enregistrats.

5.2 Obtenir versions antigues

Només hem vist una petita part de la funcionalitat d'un sistema de control de versions. Una de les característiques importants és que el servidor recorda precisament tota la història de versions i pot obtenir una còpia exacta del projecte en qualsevol moment del temps.

Per comprovar-ho situeu-vos al directori `/tmp` del vostre computador i demaneu a `subversion` una còpia antiga del projecte fent:

```
$ cd /tmp
$ svn --username 'ramon.llull' checkout -r3 http://escriny3.epsem.upc.edu/svn/<E> vantiga
```

us hauria d'haver aparegut un directori de nom `vantiga` en el que hi ha reflectida una de les versions antigues del vostre projecte. Quan ho hagueu comprovat torneu al vostre directori habitual de treball.

5.3 Emacs suporta subversion

`Emacs`, l'editor de referència, coneix quan un fitxer que s'està editant està sota el control de subversion. Ho notareu per que a la línia d'informació, sota la pantalla hi surt un missatge semblant a `SVN:4`, que indica que esteu editant la versió 4 d'aquest fitxer.

Podeu fer commit del que esteu editant directament des d'`emacs` fàcilment. Simplement cal que premeu la combinació de tecles `CTRL-X V V`. Veureu que s'obre una subfinestra a sota: és per escriure el comentari del commit. Escriviu una nota breu amb els canvis que heu fet en aquesta finestra i acabeu amb `CTRL-C CTRL-C`. Automàticament `emacs` farà el commit i pujarà la nova versió al servidor. Amb el temps anireu descobrint més funcionalitat d'`emacs` relacionades amb el control de versions.

6 Continueu la feina

Tasca 14 Ara, sense tenir un guió tant preestablert, continueu la implementació de la classe Fraccio. Treballeu en paral.lel tantes persones com hi hagi a l'equip, cadascuna amb el seu computador i feu la següent feina:

1. Implementeu el mètode __sub__()
2. Afegiu i implementeu un mètode __repr__() que permeti escriure bé una fracció.
3. Afegiu els mètodes __mul__() i __div__()
4. Afegiu el mètode __neg__() que implementa el canvi de signe.
5. Creeu un altre fitxer de nom `fraccio.txt` i, en format `RestructuredText`, hi documenteu la classe Fraccio, incloent doctests a tall d'exemple. (Sí! es poden escriure doctests en fitxers a banda!)

Intenteu treure el màxim profit possible de `subversion` per tal que cadascú pugui treballar pel seu compte. No pugeu mai coses incorrectes o inacabades, treballeu sempre completant petits blocs (una implementació, la documentació d'un mètode, etc.) i pujant aquesta aportació tant aviat com l'hagueu provada.

6.1 Lliurables

L'ús d'aquestes eines s'estendrà a la resta de pràctiques del curs i, per tant, s'avaluarà conjuntament amb aquestes pràctiques. No obstant, es demana que lliureu el resultat de les tasques que segueixen.

Tasca 15 Definiu amb les vostres paraules els conceptes següents,

1. cicle de vida
2. versions
3. desenvolupament concurrent
4. gestor de projectes

Tasca 16 Lliureu el contingut de la classe Fraccio completa, així com la documentació i doctests.

Tasca 17 Lliureu les captures de pantalla d'exemple d'ús de les vostres comandes svn.

References

[1] Sistema de control de versions pels estudiants. `http://escriny3.epsem.upc.edu`

[2] Pàgina web del projecte `subversion`. Visitada 4/4/2011. `http://subversion.apache.org`

[3] Pàgina web del projecte `redmine`. Visitada 18/3/2012. `http://www.redmine.org/`

[4] Pàgina web del projecte `trac`. Visitada 4/4/2011. `http://trac.edgewall.org`

[5] Ben Collins-Sussman, Brian W. Fitzpatrick i C. Michael Pilato, *Version Control with Subversion*, O'Reilly Media, juny 2004. També accessible lliurement a `http://svnbook.red-bean.com`