TP7 : Gestion de versions

CSH : Initiation au C et au shell

Première année

À la fin de ce TP vous saurez :
- Créer un dépôt svn et darcs
- Versionner un projet avec ces deux logiciels
- Collaborer à plusieurs sur un même projet

Les fichiers à utiliser tout au long du TP sont dans `/home/depot/1A/CSH/TP6/`. Tout le reste du TP se fait dans votre répertoire `~/gestionversions`.

Attention, les commandes citées dans ce TP ne sont pas toujours complètes ! À vous parfois de les compléter avec un nom de fichier si besoin.

Première partie
Subversion

★ Exercice 1: Mise en place

Créez le répertoire `~/gestionversions` et placez-vous dedans.

Faites `svnadmin create svndepot` pour créer le dépôt `svndepot`.

Utilisez la commande `svn checkout` pour obtenir une copie de travail du nom de `svncopie1` dans ce même répertoire, issu de `svndepot`.

Copiez le fichier ***hello.c*** de `/home/depot/...` dans cette copie de travail, et utilisez `svn add` pour le faire surveiller par svn. Vérifiez avec `svn status`. Enregistrez votre premier commit avec `svn commit -m"ajout de hello.c"`.

Vérifiez l'historique avec `svn log`. Que constatez-vous ? Faites un `svn up`[1] suivi de `svn log`. Quelle différence y a-t-il ? Cela est dû au comportement par défaut des commandes `commit` et `log`.

★ Exercice 2: Édition dans la copie de travail

Ajoutez à la fin du `main` de ***hello.c*** cette ligne : `return 0;`

Qu'affiche `svn status` ? `svn diff` ? Committez votre changement avec un nom de commit éloquent.

Récupérez les fichiers ***Makefile*** et ***README*** de `/home/depot/...`. Faites-les surveiller par svn avec `svn add`. Quelle est la syntaxe exacte pour faire ceci ?

Committez séparément l'ajout de chacun de ces fichiers avec `svn commit FICHIER -m "message"`.

Actualisez votre copie de travail et vérifiez l'historique du dépôt. Quel est le numéro de la dernière révision ?

★ Exercice 3: Récupération d'une version ancienne du dépôt

En lisant `svn help checkout`, découvrez comment obtenir une révision ancienne d'un dépôt. Récupérez la révision 1 de `svndepot` dans une nouvelle copie de travail nommée `svncopie2`.

Votre répertoire `~/gestionversions` devrait maintenant contenir ceci :

```
|-- svncopie1
|-- svncopie2
`-- svndepot
```

Entrez dans ce nouveau répertoire. Ajoutez un commentaire au tout début de ***hello.c***, avant la ligne du `#include`. Commitez. Que se passe-t-il ? Faites en sorte de pouvoir commiter ce changement.

Avec `cat hello.c` constatez les changements effectués. Commitez.

★ Exercice 4: Provoquer un conflit

Ouvrez un deuxième terminal afin d'avoir un terminal ouvert dans chaque copie de travail.

Disons que deux personnes veulent rendre le programme ***hello.c*** un peu plus international en disant bonjour dans plusieurs langues.

1. svn a des raccourcis, par exemple `up` pour `update`, `co` pour `checkout` ou `ci` pour `commit`.

Dans un dépôt, ajoutez **sous** la ligne du `printf` la ligne : `printf("Guten tag!");`
Dans l'autre, faites de même en ajoutant : `printf("Hola!");`
Puis, provoquez un conflit en tentant de commiter indépendemment ces modifications.
Résolution : dans la copie de travail qui n'a pas pu commiter, faites un `svn up`. On vous propose plusieurs options : choisissez le choix `p` pour régler le conflit à la main. Constatez avec `svn status`.
Dans le cas présent, la fusion à effectuer consiste simplement à garder les trois messages de `printf`. Éditez convenablement le fichier ***hello.c***. Signalez à svn le conflit comme résolu avec `svn resolved hello.c` et commitez.

Deuxième partie

Darcs

★ Exercice 5: Mise en place

Renseignez d'abord votre nom et e-mail pour ne pas que Darcs vous le redemande pour chaque dépôt :

```
mkdir ~/.darcs
echo "Nom Prenom <nom.prenom@esial.uhp-nancy.fr>" > ~/.darcs/author
```

Dans ~`/gestionversions`, créez le répertoire `darcsstable` et faites `darcs init` dedans. Vérifiez avec `ls`.
Copiez-y ***hello.c*** de `/home/depot/...` et faites surveiller ce fichier par Darcs avec `darcs add`.
Que produit `darcs whatsnew` ? Et `darcs whatsnew -sl` ?
Tapez simplement `darcs record`, choisissez les éléments à enregistrer et saisissez un nom de patch. Répondez non à la question vous demandant d'entrer une description longue du patch.
Vérifiez ce que vous avez fait avec `darcs changes`.

★ Exercice 6: S'organiser avec plusieurs dépôts

Remontez dans ~`/gestionversions`.
Avec `darcs get`, créez un clone de `darcsstable` nommé `darcsdev` et placez-vous dedans.
Copiez les fichiers ***Makefile*** et ***README*** de `/home/depot/...` et faites-les surveiller par darcs.
Créez un patch pour chaque ajout de fichier, en utilisant directement `darcs record -m "nom"`.
Listez les patchs de votre dépôt pour vérifier ce que vous avez fait.
Envoyez ces nouveaux patchs au dépôt parent avec `darcs push` : vous pouvez commencer par dire "non" à tout, puis "oui" juste à un des patchs, etc.

★ Exercice 7: Ne pas tout transmettre

Nous allons voir comment n'enregistrer et ne publier que ce que vous voulez rendre public.
Dans ***hello.c***, ajoutez la ligne suivante à la fin du `main` : `return 0;`
et la ligne suivante avant le `#include` : `// TODO vérifier avec gcc -Wall`
Vérifiez la présence de ces changements avec `darcs w` [2].
Enregistrez un patch qui ne contient que l'ajout du `return 0;`. Vérifiez que vous avez réussi avec `darcs w` et `darcs changes`. Envoyez ce dernier patch à `darcsstable`.

★ Exercice 8: Correction de patches

Dans ***hello.c***, changez `main()` en `int main()` et créez pour ce changement patch nommé "meilleur prototype". Ne l'envoyez pas au dépôt stable !
Changez à nouveau la ligne `int main()` en `int main(int argc, char ** argv)` et utilisez la commande `darcs amend-record` pour modifier le dernier patch que vous avez créé. Envoyez cette nouvelle version du patch au dépôt stable.
Attention, n'utilisez `amend-record` que sur des patchs non publiés ! Pourquoi ?

★ Exercice 9: Travail en binôme : préparation

Nous allons voir comment travailler en binôme avec Darcs sans avoir besoin de créer de nouvel utilisateur ou groupe Unix pour que tout le monde commite dans le même répertoire.
Par défaut, vos dossiers personnels ne sont pas accessibles aux autres utilisateurs. Une petite préparation est nécessaire. Faites `ls -l ~/..|grep monlogin`. Vous obtenez une ligne similaire à :

2. La commande darcs, comme svn, accepte des raccourcis. Elle accepte tout préfixe non ambigü, par exemple `w` pour `whatsnew`, `rec` pour `record` ou `rev` pour `revert`.

```
drwx------  42 monlogin esial1 4096 mar 24 10:00 monlogin
```

Faites `chmod go+x ~` pour permettre à tout le monde de traverser votre répertoire personnel :

```
drwx--x--x  42 monlogin esial1 4096 mar 24 10:00 monlogin
```

Mettez-vous en binôme. Si `xyz` est le login de votre binôme, faites `ls -l ~xyz/gestionversions/`. Si vous pouvez lister le contenu de ce répertoire, c'est bon.

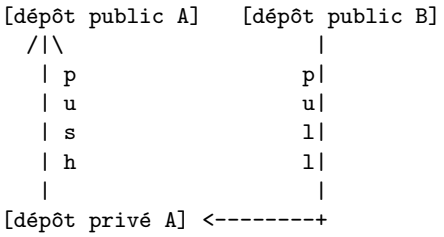
★ **Exercice 10: Mise en place des dépôts**

Parmi le binôme, attribuez-vous les rôles A et B. Vous allez mettre en place un nouveau projet. Le principe est que chaque personne travaille sur un dépôt privé et expose un dépôt public à son binôme :

1. A crée dans `~/gestionversions` un répertoire `darcscourses` et y initialise un dépôt darcs (dépôt public A)
2. A crée un clone `darcscoursesprive` de ce répertoire (dépôt privé A)
3. B crée un clone de `~A/gestionversions/darcscourses` dans son `~/gestionversions`, nommé également `darcscourses` (dépôt public B)
4. B crée un clone `darcscoursesprive` tout comme A à l'étape 2. (dépôt privé B)

Au total quatre dépôts vont servir. Les règles du jeu sont :
- on ne travaille que dans son dépôt privé
- chaque personne ne voit de l'autre que le dépôt public
- on récupère les changements de l'autre avec `pull` de son dépôt public
- on publie ses changements avec `push` vers son dépôt public

Du point de vue de la personne A :



★ **Exercice 11: Travail en binôme** Vous allez travailler sur un fichier ***listecourses*** dans lequel vous mettez deux sections, "aliments" et "autres", et un article à acheter par ligne, par exemple :

```
aliments
========
pommes
lait
fromage

autres
======
ampoule
cartes postales
```

Attention, Darcs se rappelle automatiquement de l'adresse du dernier dépôt avec lequel vous avez communiqué, donc vous devez donnez aux commandes le chemin du dépôt avec lequel vous voulez communiquer, ie `darcs push ../darcscourses/` et `darcs pull ~monbinome/gestionversions/darcscourses`.

Utilisez `darcs changes` et `cat listecourses` fréquemment pour vérifier ce qui se passe.

Reproduisez le scénario suivant :

1. B crée le fichier ***listecourses***, et un premier patch l'ajoutant. B publie ce patch.
2. A récupère ce patch.
3. B et A ajoutent quelques articles : B dans la partie "aliments", A dans la partie "autres". Ils finissent par synchroniser leurs dépôts.
4. B le récupère.
5. B crée un patch ajoutant les articles "stylos" et "papier" à la fin de la section "autres", et le publie.
6. A le récupère, puis annule cette récupération avec `darcs unpull`.

Maintenant, on continue comme si A n'était pas au courant de l'existence du dernier patch de B :

1. A ajoute "gomme" et "ciseaux" à la fin de la section "autres"
2. A crée un patch et le rend public

3. B récupère ce patch

Que se passe-t-il ?

1. B édite le fichier afin intégrer tous les changements, et crée un patch qui enregistre cette édition.
2. B publie le patch et A récupère le tout.

★ **Exercice 12: Travail avec un seul dépôt public**

Voyons comment faire pour gérer un projet similaire au précédent, de sorte qu'une seule dépôt soit public. Par exemple, on décide que le dépôt de référence du projet est situé à `~A/gestionversions/darcscourses/`, sachant que seul A peut écrire dedans. Comme A se voit confier le rôle de "garder" le dépôt du projet, A doit approuver ou désapprouver les patchs qu'on lui propose.

Lisez l'aide de `darcs send` et `darcs apply`.

Faites en sorte que B envoie un patch par courrier électronique à A et que A l'applique d'abord à son dépôt privé, puis le propage à son dépôt public [3].

Troisième partie

Retour sur svn et darcs

★ **Exercice 13:** Nouvelles étapes du scénario de l'exercice 10 :

1. A ajoute un article dans la section "autres", crée un patch et le rend public. B récupère ce patch.
2. A change d'avis et utilise `darcs rollback` pour créer un patch inverse de ce dernier patch, et le rend public. Attention à ce que `rollback` demande !

Dans quelles situations `unpull` et `rollback` servent-ils ?

★ **Exercice 14: Rendre un projet avec svn et darcs**

En utilisant `svn export` (lisez l'aide) et la commande `tar czf projet.tar.gz projet` qui crée une archive ***projet.tar.gz*** à partir d'un répertoire ***projet***, créez une archive de la dernière version de votre dépôt svn.

Quelle est la principale différence entre `svn checkout` et `svn export` ?

Lisez l'aide de `darcs dist` pour en faire de même avec un de vos dépôt Darcs.

★ **Exercice 15:** Lisez l'aide de `darcs unrecord`. En quoi cette commande diffère-t-elle de `darcs amend-record` ? Quelle est la différence entre `unpull` et `unrecord` ?

★ **Exercice 16: Checkouter un dépot SVN par le web**

Placez-vous dans `/tmp` .

Cherchez avec un moteur de recherche comment obtenir la version svn du "svn book".

Checkoutez le dépôt trouvé. Affichez l'historique (utilisez `svn log | less`).

★ **Exercice 17: Cloner un dépot Darcs par le web**

Placez-vous dans `/tmp` .

Cherchez avec un moteur de recherche comment obtenir la version darcs du programme Rubber (sa description est "rubber, a wrapper for LaTeX").

Dans le dépot cloné, listez les tags (voir l'aide de `darcs show`). Comment obtenir un dépot correspondant à un tag donné ? Créez un dépot correspondand au tag `1.0` avec `darcs get`.

3. le flag `-o` de `darcs send` sera utile