Logiciels de gestion de versions

1 Échauffement

1. Pourquoi utiliser des logiciels de gestion de versions ?
2. Quels sont les principaux outils de gestion de versions, expliquez en quoi ils diffèrent ?
3. Citez quelques services en ligne hébergeant des dépôts svn, mercurial (hg) ou git. Citez leurs avantages, inconvénients et le cas échéant, leurs coût.
4. Clonez un dépôt hg, par exemple :

```
hg clone https://bitbucket.org/sjl/dotfiles
```

5. Réalisez quelques modifications dans plusieurs fichiers du dépôts cloné.
6. Visualisez la liste des fichiers modifiés
7. Visualisez les modifications effectuées
8. Créez une nouvelle branche
9. `Committez` vos modifications.

```
Principales commandes de mercurial :

hg init         # Créer un dépôt vide
hg clone URL    # Cloner un dépôt existant
hg add          # Ajouter un fichier
hg commit       # Faire un commit
hg pull         # Envoyer les changesets sur un dépôt
hg push         # Se synchroniser avec un dépôt
hg log          # Consulter l'historique
hg status       # Consulter l'état du répertoire de travail
hg diff         # Consulter les différences
hg branch       # Créer une branche
hg tag          # Créer un tag
hg merge        # Réaliser un merge

hg help         # Consulter l'aide

# Consulter l'aide au sujet d'une commande ou d'un sujet
hg help CMD
hg CMD --help
```

1. Donner une version `git` des commandes suivante et expliquer la différence avec `hg` le cas échéant:

```
hg pull
hg merge <branch>
hg shelve
hg outgoing
hg incoming
```

2 Mercurial (hg)

> ***Note***
>
> Exercice noté, à réaliser sous linux, seul ou en binôme.

Les paquets `mercurial` `make` et `rst2pdf` sont à installer

```
$ apt-get install mercurial make rst2pdf
```

Le premier dépôt cloné dans le scénario contient un template de rapport au format `restructuredText`, un makefile et un README pointant la documentation sur le format `restructuredText`. Le rapport sera écrit dans ce format pour les besoins de l'exercice et compilé en PDF grâce à la commande :

```
$ make
```

Le rapport et les diagrammes sont a rendre par mail **au format PDF** accompagné d'une archive ZIP contenant tous les dépôts utilisé dans cet exercice.

2.1 Partie notée

1. Réalisez le scénario décrit en annexe avec `hg` **en ligne de commande**. Détaillez les commandes utilisées pour chaque étape.
2. A quoi servent les extensions mercurial suivantes : `rebase`, `shelve`, `transplant`, `mq`. Dans quels cas pourriez vous les utiliser ?
3. À l'aide d'un diagramme d'activité décrivez
    1. le processus à suivre lorsqu'un utilisateur souhaite récupérer des modifications sur le dépôt de référence.
    2. le processus à suivre lorsqu'un utilisateur ayant le droit d'écriture sur le dépôt de référence souhaite y mettre à disposition des modifications qu'il y aurait apporté.

2.2 Partie non notée

1. Lorsque vous aurez terminé, utilisez la commande `hg serve` pour permettre a vos voisin de cloner votre dépôt
2. Trouvez un moyen de déclencher un script/programme à chaque commit dans un dépôt `hg`. Utilisez ce moyen pour déclencher la commande suivante à chaque commit : `hg manifest | wc -l`
3. Utilisez TortoiseHg pour consulter l'historique de ce dépôt
4. Adaptez et réalisez le scénario avec `git`

3 Annexe

Note

Pour les besoins de ce TP, l'infrastructure client-serveur des dépôt sera simulé sur vos machines.

La simulation se fera :

- Pour le dépôt de référence : en créant un dépôt vide dans votre répertoire de TP
- pour le dépôt des utilisateurs simulés en clonant le dépôt décrit précédemment dans votre répertoire de TP

Dans l'ensemble du sujet, les lettres `XXX` sont à remplacer par le trigramme composé de la première lettre de votre prénom et des deux première lettres de votre nom. Si vous travaillez en binôme, faites de même avez vos deux trigrammes de la façon suivante: `XXX-YYY`

Dans tous les cas, les messages de commit doivent refléter ce qui à été fait dans le commit. Pas besoin de faire une dissertation, mais le message doit être clair et concis (Exemple : "Updated README.rst", "Fixed segfault due to <...>", ...)

Le rapport doit comprendre une section par étape du scénario. Pour chaque étape :

- faire référence au numéro de l'étape (exemple de titre de section : `Étape 1`)
- lister la ou les commandes exécutée pour l'étape
- lister le résultat des commandes (si celle-ci sont courtes)
- comprendre un descriptif répondant aux questions posées et/ou expliquant le choix des options de vos commandes

Pour les commandes, suivez ce modèle :

```
<username>$ hg status
A fichier.txt
M README.rst
```

en remplaçant `<username>` par le nom de l'utilisateur qui réalise la commande.

Vous êtes le chef d'une équipe composée de **Aurélie**, **Clément** et **Julien**, ainsi que d'un contributeur extérieur à l'équipe : **Loïc**. Votre projet est de réaliser le rapport de déroulement de ce scénario en suivant l'évolution du rapport avec votre outil de gestion de version favori : `hg`

Note

Aucune commande ne devra être exécutée pour ce TP dans le dépôt de référence. c'est le dépôt qu'utilise les membres de l'équipe pour synchroniser leur travail.

Néanmoins, il est possible (et conseillé) d'y jeter un œil où d'y exécuter des commande `read-only` (`hg log` par exemple) pour bien comprendre le fonctionnement de l'outil.

1. **Le chef d'équipe** crée le dépôt de référence de l'équipe :

```
~ $ mkdir TP-VCS
~ $ cd TP-VCS
~/TP-VCS $ hg clone --noupdate https://bitbucket.org/nicph/tp-vcs-template ref
~/TP-VCS $ ls
ref
```

2. **Aurélie** `clone` le dépôt de référence :

```
~/TP-VCS $ hg clone ref Aurelie
~/TP-VCS $ ls
Aurelie ref
```

3. **Aurélie** modifie le fichier `.hg/hgrc` de son clone pour y renseigner son nom d'utilisateur.
4. **Aurélie** crée un fichier `AUTHORS.txt` et y ajoute votre(vos) nom(s) et prénom(s)
5. Après avoir ajouté les fichiers nécessaires à la liste des fichiers suivis par mercurial, **Aurélie** `commit` et `push` ses modifications sur le dépôt de référence.
6. **Clément** `clone` le dépôt de référence.
7. **Clément** modifie le fichier `.hg/hgrc` de son clone pour y renseigner son nom d'utilisateur.
8. **Clément** ajoute la date du jour au fichier `README.rst` et les noms/prénoms de votre binôme.
9. **Clément** `commit` et `push` ses modifications sur le fichier `README.txt`.
10. **Aurélie** se synchronise et crée une nouvelle branche nommée `XXX-b1`.
11. **Aurélie** `push` la nouvelle branche sur le dépôt de référence. La commande échoue. Expliquez pourquoi, et comment résoudre le problème.
12. **Clément** se synchronise avec la branche `XXX-b1`.
13. **Clément** modifie le fichier `rapport.rst` en remplacant `Nom1 Prenom1, Nom2 Prenom2` par votre(vos) nom(s) et prénom(s) et en y mettant à jour le titre et en intégrant les notes prise par vos soins jusqu'à maintenant.
14. **Clément** génère le fichier PDF avec la commande `make` pour visualiser le rendu.
15. **Clément** affiche l'état de son dépôt (`hg status`). Quel est le résultat ?
16. **Clément** `commit` uniquement les modification apportées au fichier `rapport.rst`.
17. **Clément** affiche l'état de son dépôt. Quel est le résultat ? Pourquoi ?
18. **Clément** ne souhaite pas pas ajouter le fichier PDF au dépot par accident. Trouvez un moyen pour que mercurial ignore ce type de fichier dans ce dépot.
19. **Clément** `commit` le fichier contenant cette liste.
20. **Clément** `push` sur dépôt de référence l'ensemble des changesets qui on été créés.

    Maintenant, votre répertoire devrait ressembler à ceci :

```
TP-VCS/
|-- Aurelie/
|   |-- .hg/
|   |-- Makefile
|   |-- README.rst
|   \-- rapport.rst
|-- Clement/
```

```
|   |-- .hg/
|   |-- ?????????
|   |-- Makefile
|   |-- README.rst
|   \-- rapport.rst
\-- ref/
    \-- .hg/
```

21. **Julien** `clone` le dépôt de référence.
22. **Julien** modifie le fichier `.hg/hgrc` de son clone pour y renseigner son nom d'utilisateur.
23. **Julien** se met à jour sur la branche `XXX-b1`.
24. **Julien** Modifie le fichier `rapport.rst` (avec vos notes) et met à disposition ces modifications sur le dépôt de référence.
25. **Clément** synchronise son clone avec le dépôt de référence et met son code à jour avec la branche `XXX-b1`.
26. **Clément** étudie l'historique (les `logs`) (n'hésitez pas à détailler et tester les options possibles).
27. **Clément** regarde précisément la dernière modification effectuée par **Julien**. (le diff, la différence ligne par ligne). Donner plusieurs commandes pour obtenir ce résultat.
28. **Clément** modifie dans son dépôt le fichier `rapport.rst` (avec vos notes) mais **NE MET PAS** à disposition ces modifications sur le dépôt de référence (`commit` mais ne `push` pas).
29. **Julien** consulte l'historique de son clone.
30. **Julien** modifie dans son dépôt le fichier `rapport.rst` avec les notes sur la commande de consultation d'historique.
31. **Julien** `commit` (mais ne `push` pas).
32. **Clément** met à disposition ses modifications sur le dépôt de référence.
33. **Julien** met à disposition ses modifications sur le dépôt de référence. Quel sont les problèmes rencontrés ? Expliquez comment vous résolvez ces problèmes et les choix que vous réalisez. Au final, **Julien** s'arrange pour que le rapport soit cohérent avec la chronologie de ce scénario.
34. **Clément** Après s'être synchronisé, ajoute un fichier `LICENSE.txt` contenant une license de votre choix et `commit` sa modification (pas de `push`).
35. **Clément** ajoute un fichier `ROADMAP.txt` et `commit` sa modification (pas de `push`).
36. **Clément** `push` l'ensemble de ses modifications. (Les problèmes éventuellement rencontrés doivent être résolus pour permettre le `push`).
37. **Julien** met à jour le fichier `rapport.rst` (avec vos notes) `commit` sa modification (pas de `push`).
38. **Julien** souhaite mettre à disposition ses modifications. **Julien** souhaite également savoir si des changeset on été ajouté au dépôt de référence avant de réaliser son push. Quelle commande(s) peut-elle être utilisée par **Julien** ?
39. **Julien** Constate qu'il n'est pas synchronisé avec le dépôt de référence. **Julien** souhaite également éviter de réaliser un `merge` pour obtenir un historique plus linéaire. Quelle commande(s) peut remplacer `merge` dans cette situation ? Quelle est la différence avec la commande `merge` ?
40. **Julien** `push` l'ensemble de ses modifications.
41. Après s'être synchronisé, **Clément** déplace `rapport.rst` et `Makefile` dans un sous-dossier `src` en s'assurant que `hg` prenne bien en compte ces déplacement.

42. **Clément** `push` ses modifications.
43. **Julien** se synchronise avec le dépôt de référence.
44. **Julien** affiche l'historique du fichier (`rapport.rst`).
45. **Aurélie** se synchronise et ajoute un `tag XXX-v1.0` sur le dernier changeset de la branche `XXX-b1`.
46. **Aurélie** crée une nouvelle branche nommée `XXX-b2` à partir du changeset pointé par le tag `XXX-v1.0` et la met à disposition de l'équipe.
47. **Clément** bascule sur la branche `XXX-b2`.
48. **Clément** met à jour le fichier `rapport.rst` (avec vos notes) et les met les modifications a disposition de l'équipe.
49. **Julien** bascule sur la branche `XXX-b2`.
50. **Julien** met à jour le fichier `rapport.rst` (avec vos notes) et les met les modifications a disposition de l'équipe.
51. **Clément** bascule sur la branche `XXX-b1`.
52. **Clément** fait une correction dans le rapport (vous trouverez bien une faute d'orthographe à corriger ...) et met sa modification a disposition de l'équipe.
53. **Clément** consulte les différences entre sa version courante et le tag `XXX-v1.0`.
54. **Aurélie** ajoute le `tag XXX-v1.1` sur la version courante de la branche `XXX-b1`.
55. **Aurélie** `merge` les dernières modifications liés au tag `XXX-v1.1` sur la branche `XXX-b2` et met ses modification à disposition de l'équipe.
56. **Aurélie** souhaite visualiser l'historique complet du dépôt. Quelle commande pourra utiliser **Aurélie** ? Quelles options sont nécessaire/possible pour visualiser le graphe, le résumé de chaque changeset et les fichiers modifiés par chaque changeset ?
57. **Loïc** souhaite contribuer au projet, mais n'a qu'un accès en lecture au dépôt de référence. **Loïc** `clone` le dépôt de référence sur sa machine et bascule sur la branche `XXX-b2`.
58. **Loïc** modifie le fichier `.hg/hgrc` de son clone pour y renseigner son nom d'utilisateur.
59. **Loïc** n'a pas les droits pour `push` ses modifications sur dépôt de référence. Pour simuler celà, supprimez la section `[paths]` du fichier `.hg/hgrc` de son clone.
60. **Loïc** est voyant, et saisi dans le rapport la section concernant l'étape suivante.
61. **Loïc** crée un patch a partir de ses modifications.
62. **Aurélie** obtient le patch soumit par **Loïc** au projet, l'analyse et l'applique sur la branche `XXX-b2`.
63. **Loïc** se resynchronise par rapport à dépôt de référence pour pouvoir continuer sa contribution.
64. **Aurélie** saisi les dernières notes et met ses modification à disposition de l'équipe. **Aurélie** génère le PDF final.