Projet Shell : Gestion de versions d'un fichier
Système et programmation système

Université de Franche-Comté – UFR Sciences et Technique
Licence Informatique – 2è année
2022 – 2023

Le but de ce projet est de réaliser un utilitaire de gestion de versions appelé `version.sh`. Comme son nom l'indique, `version.sh` est implémenté en langage shell. Son rôle est de gérer les différentes versions d'un fichier qu'on va pouvoir sauvegarder au fur et à mesure. Hormis avec l'option `--help` détaillée par la suite, l'utilitaire s'utilise toujours de la même manière :

```
$ ./version.sh <command> FILE [OPTION]
```

`<command>` représente une commande de l'utilitaire, `FILE` est le fichier concerné, tandis qu'`OPTION` représente l'éventuelle option de la commande. Nous proposons d'implémenter les commandes suivantes : `add`, `rm`, `commit`, `diff`, `checkout`, `log`, `reset` et `amend`. Le détail de ces commandes sera expliqué par la suite.

Spécification

Pour les fichiers versionnés, c'est-à-dire gérés par le gestionnaire de versions, les différentes versions seront sauvegardées dans un répertoire caché nommé `.version`. La structure de ce répertoire est imposée par le projet. Le répertoire `.version` doit toujours se trouver **dans le même répertoire** que le fichier versionné.

Dans le répertoire `.version`, pour chaque fichier versionné, on trouvera :
- La première version du fichier, avec l'extension `.1` ;
- La dernière version du fichier, avec l'extension `.latest` ;
- Un ensemble de patchs entre les versions successives, avec l'extension `.N` où `N` correspond à la $\texttt{N}^{\text{e}}$ version du fichier.

Par exemple, si on a déjà 3 versions successives du fichier `example.txt`, le répertoire `.version` contient :
- `example.txt.1` qui est une copie de la première version du fichier ;
- `example.txt.2` et `example.txt.3` qui sont des patchs entre les versions 1 et 2, et 2 et 3 respectivement ;
- `example.txt.latest` qui est une copie de la dernière version du fichier.

Donc, `example.txt.1` et `example.txt.latest` sont des copies du fichier à un instant donné, tandis que les 2 autres fichiers sont des patchs.

Le répertoire `.version` doit toujours être cohérent par rapport à cette spécification. Il est nécessaire de faire toutes les vérifications utiles pour ne pas le rendre incohérent.

Le script devra pouvoir être appelé depuis n'importe quel répertoire, pour n'importe quel fichier dans le système de fichiers (en utilisant des chemins qui les désignent). Vous pourrez pour cela vous aider des commandes `dirname(1)` et `basename(1)`. Remarque : on supposera que tous les noms des fichiers et des répertoires contiennent uniquement des lettres, des chiffres, des underscores (_), des tirets (-) et des points (.).

Pour chaque commande `<command>`, le script devra vérifier la validité des arguments qui lui sont transmis (nombre d'arguments et valeurs des arguments). En cas d'erreur, il faut afficher, sur la sortie d'erreur standard, un message indiquant l'origine de l'erreur, inviter l'utilisateur à consulter l'aide avec le message ci-dessous, puis sortir.

```
Enter "./version.sh --help" for more information.
```

Il faudra notamment vérifier que le fichier `FILE` est un fichier ordinaire existant avec la permission de lecture, et que la commande existe :

```
$ ./version.sh add unknown.txt 'Initial commit'
Error! 'unknown.txt' is not a regular file or read permission is not granted.
Enter "./version.sh --help" for more information.
$
$ ./version.sh unknown dir/example.txt
Error! This command name does not exist: 'unknown'
Enter "./version.sh --help" for more information.
```

Pour chaque appel à l'utilitaire `version.sh`, quand cela a un sens, il faudra notamment :

— vérifier la présence du répertoire `.version` ;

— vérifier si le fichier `FILE` est sous contrôle du gestionnaire de version.

Il est très important de mettre un maximum de messages d'information à destination de l'utilisateur pour que celui-ci puisse vérifier le bon fonctionnement ou les erreurs survenues. Les messages seront **en anglais**. Des exemples sont donnés dans les questions qui suivent, mais tous les messages ne sont pas donnés. Néanmoins, vous pouvez faire les hypothèses suivantes :

— l'utilisateur ne modifie pas "directement" le contenu du répertoire `.version`, c'est à dire sans utiliser le script `version.sh` ;

— si le script `version.sh` arrive à créer au moins un fichier dans le répertoire `.version`, il réussira à en créer d'autres.

Réalisation

Exercice 1 : `diff(1)` et `patch(1)`

Si vous ne l'avez pas déjà fait, faire l'exercice 19 du carnet de travaux libres, portant sur les commandes `diff(1)` et `patch(1)`.

Remarque : quand le fichier texte sous contrôle du gestionnaire de versions est un gros fichier avec relativement peu de différences entre les différentes versions, sauvegarder les patchs entre les différentes versions plutôt que les versions elles-mêmes permet de diminuer le poids des fichiers utilisés.

Exercice 2 : Option `--help`

Le script `version.sh` doit disposer d'une option `--help` qui affiche le texte suivant :

```
$ ./version.sh --help
Usage:
    ./version.sh --help
    ./version.sh <command> FILE [OPTION]
    where <command> can be: add amend checkout|co commit|ci diff log reset rm

./version.sh add FILE MESSAGE
    Add FILE under versioning with the initial log message MESSAGE

./version.sh commit|ci FILE MESSAGE
    Commit a new version of FILE with the log message MESSAGE

./version.sh amend FILE MESSAGE
    Modify the last registered version of FILE, or (inclusive) its log message

./version.sh checkout|co FILE [NUMBER]
    Restore FILE in the version NUMBER indicated, or in the
    latest version if there is no number passed in argument

./version.sh diff FILE
    Displays the difference between FILE and the last committed version

./version.sh log FILE
    Displays the logs of the versions already committed

./version.sh reset FILE NUMBER
    Restores FILE in the version NUMBER indicated and
    deletes the versions of number strictly superior to NUMBER

./version.sh rm FILE
    Deletes all versions of a file under versioning
```

Question 2.1 Écrire une fonction `help` qui affiche le message d'aide indiqué ci-dessus.

Si l'option `--help` est utilisée, elle doit être la seule présente.

Question 2.2 Afficher l'aide et terminer le script si l'option `--help` est la seule présente.

Exercice 3 : Commande `add`

```
Usage : ./version.sh add FILE MESSAGE
    Add FILE under versioning with the initial log message MESSAGE
```

Question 3.1 Implémenter la commande `add` qui met un fichier sous contrôle du gestionnaire de version, et sauvegarde la version 1, qui est également la dernière version dans ce cas. Ne vous préoccupez pas du message de log dans un premier temps. La gestion de ces messages de log est présentée dans l'exercice 8 "Gestion des messages de log et commande `log`".

```
$ ./version.sh add dir/example.txt 'Initial version'
Added a new file under versioning: 'example.txt'
```

Exercice 4 : Commande `rm`

```
Usage : ./version.sh rm FILE
    Deletes all versions of a file under versioning
```

Question 4.1 Implémenter la commande `rm` qui supprime toutes les versions d'un fichier sous contrôle. Demander une confirmation de la part de l'utilisateur. Supprimer le répertoire `.version` s'il est vide.

```
$ ./version.sh rm dir/example.txt
Are you sure you want to delete 'example.txt' from versioning ? (yes/no) no
Nothing done.
```

```
$ ./version.sh rm dir/example.txt
Are you sure you want to delete 'example.txt' from versioning ? (yes/no) yes
'example.txt' is not under versioning anymore.
```

Exercice 5 : Commande `commit`

La commande `commit` dispose d'un alias égal à `ci`.

```
Usage : ./version.sh commit|ci FILE MESSAGE
    Commit a new version of FILE with the log message MESSAGE
```

Question 5.1 Implémenter la commande `commit` qui ajoute une nouvelle version d'un fichier sous contrôle. Indiquer, dans le message destiné à l'utilisateur, le numéro de la version qui vient d'être «commitée». Si le fichier courant est identique à la dernière version, on ne commitera rien. (indice : `cmp(1)`). Ne vous préoccupez pas du message de log dans un premier temps. La gestion de ces messages de log est présentée dans l'exercice 8 "Gestion des messages de log et commande `log`".

```
$ ./version.sh commit dir/example.txt 'Add an update'
Committed a new version: 2
```

Exercice 6 : Commande `diff`

```
Usage : ./version.sh diff FILE
    Displays the difference between FILE and the last committed version
```

Question 6.1 Implémenter la commande `diff` qui affiche la différence entre la version actuelle du fichier et la dernière version commitée. Le format de la différence est le format unifié.

```
$ ./version.sh diff dir/example.txt
--- dir/.version/example.txt.latest 2023-03-02 15:58:01.399937873 +0100
+++ dir/example.txt 2023-03-02 16:00:01.507363426 +0100
@@ -1,3 +1,4 @@
 line 1
 line 2
 line 3
+line 4
$
```

Exercice 7 : Commande `checkout`

La commande `checkout` dispose d'un alias égal à `co`.

```
Usage : ./version.sh checkout|co FILE [NUMBER]
    Restore FILE in the version NUMBER indicated, or in the
    latest version if there is no number passed in argument
```

Question 7.1 Implémenter la commande `checkout` qui restaure le fichier dans la version indiquée par l'argument `NUMBER`, ou dans la dernière version "commitée" si l'argument `NUMBER` est absent. S'il est présent, l'argument `NUMBER` doit être un entier supérieur ou égal à 1, et correspondre à une version qui existe. Pour les versions autres que la dernière, vous pouvez partir de la première version et appliquer les patchs successivement.

```
$ ./version.sh checkout dir/example.txt
Checked out to the latest version

$ ./version.sh checkout dir/example.txt 1
Checked out version: 1
```

Exercice 8 : Gestion des messages de log et commande `log`

```
Usage : ./version.sh log FILE
    Displays the logs of the versions already committed
```

Pour chaque version (cf. commandes `add`, `commit` et `amend`), on associe un commentaire d'une seule ligne qui sera sauvegardé dans un fichier `.log` dans le répertoire `.version`. Par exemple, le log pour le fichier `example.txt` sera sauvegardé dans `example.txt.log`. Le commentaire, entouré de simples quotes, sera préfixé par la date et l'heure du jour au format RFC-5322 (Indice `date(1)`). Les espaces blancs en début et en fin de commentaire doivent être supprimés avant sa mémorisation dans le fichier `.log`.

Question 8.1 Vérifier que le commentaire n'est pas vide, et qu'il est bien sur une seule ligne.

```
$ ./version.sh commit dir/example.txt '   Add an another update   '
Committed a new version: 3
```

Question 8.2 Implémenter la commande `log` qui affiche le log des versions déjà commitées. Chaque ligne du fichier de log sera précédé par le numéro de la ligne et par un caractère deux-points (:).(indice : `nl(1)`).

```
$ cat dir/.version/example.txt.log
Thu, 02 Mar 2023 11:54:27 +0100 'Initial version'
Thu, 02 Mar 2023 11:55:45 +0100 'Add an update'
Thu, 02 Mar 2023 11:56:07 +0100 'Add an another update'

$ ./version.sh log dir/example.txt
  1 : Thu, 02 Mar 2023 11:54:27 +0100 'Initial version'
  2 : Thu, 02 Mar 2023 11:55:45 +0100 'Add an update'
  3 : Thu, 02 Mar 2023 11:56:07 +0100 'Add an another update'
```

Exercice 9 : Commande `reset`

```
Usage : ./version.sh reset FILE NUMBER
    Restores FILE in the version NUMBER indicated and
    deletes the versions of number strictly superior to NUMBER
```

Question 9.1 Implémenter la commande `reset` qui restaure le fichier dans la version indiquée par l'argument `NUMBER`, et qui supprime les versions de numéro strictement supérieur à `NUMBER`. Les log des versions supprimées doivent également être supprimés. L'argument `NUMBER` doit être un entier supérieur ou égal à 1, et correspondre à une version qui existe. Un `reset` à la dernière version "commitée" revient à faire un `checkout` à cette dernière version. Si au moins une version est supprimée, il faut demander une confirmation à l'utilisateur.

```
$ ./version.sh log dir/example.txt
  1 : Thu, 02 Mar 2023 11:54:27 +0100 'Initial version'
  2 : Thu, 02 Mar 2023 11:55:45 +0100 'Add an update'
  3 : Thu, 02 Mar 2023 11:56:07 +0100 'Add an another update'

$ ./version.sh reset dir/example.txt 3
Checked out to the latest version

$ ./version.sh reset dir/example.txt 2
Are you sure you want to reset 'example.txt' to version 2 ? (yes/no) yes
Reset to version: 2

$ ./version.sh log dir/example.txt
  1 : Thu, 02 Mar 2023 11:54:27 +0100 'Initial version'
  2 : Thu, 02 Mar 2023 11:55:45 +0100 'Add an update'
```

Exercice 10 : Commande `amend`

```
Usage : ./version.sh amend FILE MESSAGE
    Modify the last registered version of FILE, or (inclusive) its log message
```

Question 10.1 Implémenter la commande `amend` qui permet de modifier la dernière version "commitée" du fichier `FILE`, ou (inclusif) de modifier son message de log.

```
$ cat dir/example.txt
line 1
line 2
line 3

$ echo 'line 4' >> dir/example.txt

$ ./version.sh commit dir/example.txt 'Add line 4'
Committed a new version: 3

$ ./version.sh log dir/example.txt
  1 : Thu, 02 Mar 2023 11:54:27 +0100 'Initial version'
  2 : Thu, 02 Mar 2023 11:55:45 +0100 'Add an update'
  3 : Thu, 02 Mar 2023 15:09:44 +0100 'Add line 4'

$ echo 'line 5' >> dir/example.txt

$ ./version.sh amend dir/example.txt 'Add line 4 & 5'
Latest version amended: 3

$ ./version.sh log dir/example.txt
  1 : Thu, 02 Mar 2023 11:54:27 +0100 'Initial version'
  2 : Thu, 02 Mar 2023 11:55:45 +0100 'Add an update'
  3 : Thu, 02 Mar 2023 15:10:40 +0100 'Add line 4 & 5'

$ cat dir/.version/example.txt.latest
line 1
line 2
line 3
line 4
line 5
```

Évaluation

La qualité de l'architecture du script (factorisation, découpage en fonctions, ...) sera prise en compte. Les choix d'implémentation doivent être justifiés à l'aide de commentaires dans le script. Chaque définition de fonction doit être précédée d'un bloc de commentaires indiquant notamment son rôle et les paramètres attendus.

Vous devrez déposer un script nommé `version.sh` dans le dépôt MOODLE prévu à cet effet avant le :

mardi 21 mars 2023 à 22h00.

Le script doit être interprétable par l'interpréteur de commandes `dash`. Dans le cas contraire, votre rendu serait considéré comme hors sujet.

Le projet est à faire en binôme, vous indiquerez dans un bloc de commentaires, en haut de votre fichier `version.sh`, juste après le shebang :

- le nom des deux membres du binôme, ainsi que vos groupes de TP respectifs ;
- une conclusion/bilan sur le produit final (ce qui est fait, testé, non fait). Pour les commandes non implémentées, le script affichera : `Not implemented` ;
- un bilan par rapport au travail en binôme ;
- toute autre remarque que vous jugerez utile.