**PROJET MODÉLISATION D'UNE BD +**

**SQL SERVEUR**

****

****

**Groupe :**

**AMROUNE** Sofiane

**CHALAUX** Florian

**GALY** Pierre

**MOREAU** Théo

### Sommaire

**Mise en place du dépôt GitHub1**

Définition et comparatif2

Utilisation et gestion d’un client Git3

**GLPI4**

Analyse5

Traitement et configuration de l’application6

**Mise en place du serveur SQL4**

Définition des besoins d’architecture du serveur de base de données5

Manipulation et paramétrage du serveur de base de données6

Accès et droits6

Tests6

Remplissage des données6

Mise en place des stockages6

Programmation des sauvegardes 6

**Mise en place du dépôt GitHub :**

1. Définition et comparatif

Aujourd'hui, la taille des projets informatiques est de plus en plus importante. Les équipes de développements comptent, ainsi, de plus en plus de membres. Tout au long de la réalisation des programmes informatiques, de nombreuses personnes "commitent" tous les jours leur code. Avec les systèmes de gestion de source traditionnels tels que CVS ou SVN, il devient de plus en plus compliqué de travailler au quotidien. Les problèmes récurrents sont :

* Les conflits de code lors des commits suite au travail de deux développeurs sur une même portion de code,
* La gestion des différentes versions de code (celle en recette et celle en cours de développement principalement),
* La difficulté de valider l'intégration de son code avec le reste du projet avant le commit.

La cause principale de ces soucis que rencontrent les grosses équipes de développements au quotidien est la centralisation du repository contenant le code source partagé. Pour cette raison, des logiciels de gestion de version distribués ont été créés. Ils gèrent les versions à l'aide d'une architecture des dépôts du code source distribuée. En effet, ces logiciels impliquent que chaque développeur a une copie locale d'un "repository" de référence ou maître avec lequel elle est synchronisée.

**À propos de la gestion de version :**

Un gestionnaire de version est un système qui enregistre l'évolution d'un fichier ou d'un ensemble de fichiers au cours du temps de manière à ce qu'on puisse rappeler une version antérieure d'un fichier à tout moment.

Il permet de répertorier ces changements et de revenir dessus si besoin est.   
De plus, il facilite le travail d'équipe en vous avertissant s'il y a des conflits (si deux personnes ont édité un même fichier en même temps).

Voici quelques exemples :

**GIT :**  un [logiciel de gestion de versions](https://fr.wikipedia.org/wiki/Logiciel_de_gestion_de_versions) [décentralisé](https://fr.wikipedia.org/wiki/Gestion_de_versions#Gestion_de_versions_d.C3.A9centralis.C3.A9e). C'est un [logiciel libre](https://fr.wikipedia.org/wiki/Logiciel_libre) créé par [Linus Torvalds](https://fr.wikipedia.org/wiki/Linus_Torvalds), auteur du [noyau Linux](https://fr.wikipedia.org/wiki/Noyau_Linux), et distribué selon les termes de la [licence publique générale GNU](https://fr.wikipedia.org/wiki/Licence_publique_g%C3%A9n%C3%A9rale_GNU) version 2. En 2016, il s’agit du [logiciel de gestion de versions](https://fr.wikipedia.org/wiki/Logiciel_de_gestion_de_versions) le plus populaire qui est utilisé par plus de douze millions de personnes.

**GNU Arch :** Arch suit l'ensemble de l'arborescence -- et pas seulement des fichiers individuels. Par exemple, on peut changer de nombreux fichiers dans une arborescence, arch peut enregistrer tous ces changements comme un ensemble de changements, plutôt que fichier par fichier. Si on renomme les fichiers ou si on réorganise l'arborescence, arch est capable d'enregistrer ces modifications, de la même manière qu'il le fait lors des modifications de fichiers.

**Mercurial**: un [logiciel de gestion de versions](https://fr.wikipedia.org/wiki/Logiciel_de_gestion_de_versions) [décentralisé](https://fr.wikipedia.org/wiki/Gestion_de_version_d%C3%A9centralis%C3%A9e) disponible sur la plupart des systèmes Unix (linux et mac OS X) et Windows. Mercurial est écrit principalement en [Python](https://fr.wikipedia.org/wiki/Python_(langage)). Il a été créé pour s'utiliser via des lignes de commandes. Toutes les commandes commencent par « hg », en référence au symbole chimique du [mercure](https://fr.wikipedia.org/wiki/Mercure_(chimie)). Ses principales caractéristiques sont[3](https://fr.wikipedia.org/wiki/Mercurial#cite_note-3), entre autres :

* Sa rapidité et sa capacité à gérer les gros projets ;
* Son utilisation sans nécessiter un serveur ;
* Son fonctionnement complètement distribué ;
* Sa robustesse dans la gestion des fichiers ASCII et binaires ;
* Sa gestion avancée des branches et des fusions ;
* Son interface web intégrée.

Comparatif :

Git :

Avantages :

* **Gestion des branches**: possibilité de travailler sur plusieurs projets en parallèle.

Les développeurs peuvent avoir des branches locales, des branches publiques. Les branches locales sont utiles lorsqu’ils travaillent individuellement sur une nouvelle fonctionnalité compliquée qui implique plusieurs fichiers. Pendant leur travail, ils peuvent faire des “pull” et ainsi mettre à jour leur version de développement avec les modifications faites par les collègues. À mesure qu’ils atteignent des objectifs intermédiaires, ils peuvent faire des commit qui resteront locaux. Lorsque la fonctionnalité est terminée, ils peuvent faire un “merge” vers la branche commune de développement (nous on utilise Master), puis “push” et tout le monde aura accès à leur code.

* **L’interface console** :
* **Algorithmes de fusion** : quand un fichier a été modifié par plusieurs personnes en même temps, Git sait s’adapter et choisir un algorithme qui fusionne intelligemment les lignes du fichier qui ont été modifiées. Si par hasard 2 personnes ont modifié en même temps la même ligne (cas rare, mais qui arrive), il y a un conflit et Git laisse des marques dans le fichier pour dire qui a modifié quoi, et vous invite à décider ce que vous gardez.
* **La rapidité :** les mises à jours et données sont fusionnées très rapidement même s’il y a eu de nombreuses modifications.
* **Le Contenu :** contrairement à d’autres logiciels de gestion, Git ne surveille pas les fichiers mais leur contenu.

Inconvénients :

* **La complexité** : Prise en main compliqué et pas facile pour n’importe qui. Logiciel fait par des développeurs pour des développeurs.
* **Le portage sous Windows**: il faut utiliser cygwin. Difficultés de connexions au serveur SSH. Interface console windows n’est pas trop courante.

Mercurial :

|  |  |  |
| --- | --- | --- |
|  | Avantages | Inconvénients |
| GIT | * Rapidité * Facilité d’installation * Gestion des branches * Algorithmes de fusion * Interface console * Contenu | * La complexité * Portage sous windows * Retour à la ligne * Très grand nombre de commandes |
| Mercurial |  |  |
| GNU arch |  |  |

**GLPI :**

1. Analyse des besoins et des utilisations de GLPI.

GLPI est une solution de Gestion Libre de Parc informatique, elle permet de gérer tous les éléments liés à l'informatique.

GLPI permet la gestion du matériel informatique, la gestion des demandes d'assistance d'utilisateurs ainsi que la gestion des licences.

GLPI est une solution open-source gratuite en application web.