INITIATION A GIT

« git : a person, especially a man, who is stupid or unpleasant »

Table des matières

A) Introduction

GIT est un système de révision de code source, créé par Linus Torvalds, le développeur finlandais, naturalisé américain, à l'origine du noyau Linux, utilisé dans tous les systèmes d'exploitation GNU/Linux (Debian, RedHat, SuSE, Ubuntu, Mint, Arch, ...).

Son but est de permettre un travail en équipe sur un projet commun, en facilitant les intégrations, les revues, les corrections inhérentes à tout projet de programmation.

Chaque contributeur a accès à deux dépôts :

➔ un dépôt « distant », où les contributeurs peuvent téléverser leurs contributions et télécharger celles des autres participants.

➔ un dépôt « local », qui reflète tout ou parti du dépôt commun, et dans lequel vous devez travailler au quotidien.

GIT impose par défaut une branche principale nommée **master** (renommée parfois en **main**), quel que soit le projet. Mais ensuite, c'est à vous de créer vos branches !

En général, on créé tout de suite une branche **dev** (développement) sur le dépôt commun, qui est la branche principale de travail pour tous les contributeurs.

Chaque développeur ramène cette branche **dev** sur son dépôt local, développe des fonctionnalités dans des **branches feature/xxx** (fonctionnalité/xxx), puis fusionne (**merge**) ses changements à la branche **dev**, avant renvoi de cette dernière au dépôt distant, qui peut avoir changé entre temps...

Ainsi le dépôt distant ne contient jamais les branches locales **feature/xxx** des contributeurs – juste les fusions de ces dernières,

accompagnées parfois d'un commentaire qui doit être le plus explicite.

Car une autre contrainte non écrite, mais largement respectée, est d'avoir toujours un journal de fusion propre et succint. Et c'est loin d'être facile quand vous travaillez en groupe… Là encore, il faut se donner des règles communes, pour éviter de rendre l'historique illisible.

Règle de travail principale sous GIT : **on ne travaille JAMAIS directement sur une branche commune** : on créé toujours une branche locale qui part d'une branche commune, et que l'on refusionne à cette dernière une fois la fonctionnalité au point.

Il n'est pas rare dans les projets courants de trouver d'autres branches : **release** (version publiée), **hotfix** (correctif rapide d'une release), **support**, etc.

Mais comprenez que GIT n'impose aucune organisation ! C'est à vous de trouver la vôtre !

En cas de conflit, qui arrive tôt ou tard, GIT présente les codes source en concurrence, et c'est au développeur de faire le tri, comme on le verra plus loin.

Enfin, un des aspects les plus déconcertant de GIT, quand on apprend l'outil, est qu'il transforme votre dossier de travail en dossier virtuel, permettant de revenir en arrière, de remiser du code, de réécrire l'historique du projet, et plein d'autres choses encore qui expliquent sa très large adoption !

Enfin, bien qu'il existe des outils graphiques intégrés aujourd'hui aux GUI de programmation, nous ne travaillerons ici qu'en ligne de commande (CLI), afin de bien comprendre et analyser ce qui se passe et pourquoi.

B) Storcka.fr

Nous prendrons ici l'exemple de la société fictive **Storcka.fr** disposant de trois développeurs (Alice, Bob et René), devant créer un site web .

Pour émuler les interactions, nous aurons besoin :

➔ de GIT (quelle surprise !)

➔ de PHP 7.4 et 8+ (version volontairement dépassées, mais qui permettront d'émuler un changement de version majeur chez un prestataire hébergeur par exemple)

➔ et d'autres outils en CLI que nous verrons plus loin.

C) Mode administrateur sous GNU/Linux

Si le super-utilisateur a un mot de passe :

```
su -l
```

Sinon

```
sudo -s
```

ou

```
sudo bash
```

D) Installation de paquets

Rappel : l'installation de paquets se fait toujours en mode administrateur.

Sous GNU/Linux et pour une Debian/Ubuntu/Mint, c'est :

```
apt update && apt dist-upgrade
apt install git tree vim meld php curl wget
```

PHP pour Debian 12 :

```
apt install -y apt-transport-https lsb-release ca-
certificates
wget -O /etc/apt/trusted.gpg.d/php.gpg
https://packages.sury.org/php/apt.gpg
echo "deb https://packages.sury.org/php/ $
(lsb_release -sc) main" | tee
/etc/apt/sources.list.d/php.list
```

PHP pour Ubuntu 22.04 :

```
apt install software-properties-common
add-apt-repository ppa:ondrej/php
```

Choix de la version pour Debian 12 et Ubuntu 22.04 :

```
apt update
apt install apache2
# sur une ligne
apt install php7.4 php7.4-
{fpm,cli,common,curl,zip,gd,mysql,xml,mbstring,json,
intl}
# sur une ligne
apt install php php-
{fpm,cli,common,curl,zip,gd,mysql,xml,mbstring,json,
intl}
# vérification de la version
php -v
```

```
# passage à php7.4 en mode auto
update-alternatives --set php /usr/bin/php7.4
# passage à une autre version en mode manuel
#update-alternatives --config php
php -v
```

GIT pour Windows

➔ À télécharger sur https://git-scm.com/downloadset.

PHP pour Windows :

➔ À télécharger sur https://windows.php.net/download/ en version ZIP / Thread Safe, et à décompresser dans *C:\PHP*.

➔ Rajoutez le chemin *C:\PHP* dans votre variable d'environnement **$PATH** pour pointer vers PHP 7.4 au début de ce TP.

E) Dossier de travail principal

Astuce : si vous êtes sous GNU/Linux, vous pouvez installer rapidement le serveur web Apache2, puis aller dans le dossier d'Apache par défaut (en général */var/www/html*, à vérifier dans les fichiers de configuration */etc/apache2*).

Créez un dossier *CoursGit* :

```
mkdir CoursGit
cd CoursGit
```

et dans ce dernier, quatre sous-dossiers *alice*, *bob*, *rene* et *le.depot.distant*.

```
mkdir alice bob rene le.depot.distant
```

E.1) Préparation des dossiers

Allez dans le dossier d'alice et créez le sous-dossier :

```
cd alice
mkdir www.storcka.fr
```

Allez dans ce sous-dossier :

```
cd www.storcka.fr
```

Créez le fichier *README.md* avec le contenu :

```
Site web : www.storcka.fr
Client : D\'Storcka
Version : 1.0
```

Initialisez le dépôt de travail :

```
git init .
```

Affichez l'état du tampon d'envoi :

```
git status
```

Remarquez que par défaut, nous sommes sur la branche **master**

Configurez les identifiants GIT de l'usager en local :

```
git config user.name Alice
git config user.email alice@storcka.fr
git config color.ui auto
```

```
git config -l
git config user.name
git config user.email
git config  color.ui
```

Remarquer la présence de la variable `init.defaultbranch` qui fixe le nom de la première branche par défaut du projet (**master** par défaut).

P.S. : nous n'utilisons pas ici l'option `--global` qui permet de fixer ces réglages pour tous les dépôts GIT créés par l'usager sur sa machine. La raison est que nous allons ici émuler les contributions de nos trois développeurs, chacun d'entre eux ayant ses propres réglages.

Ajoutez les fichiers présents :

```
git add .
```

Affichez l'état du tampon d'envoi :

```
git status
```

Créez la première révision du projet :

```
git commit -m "Début du projet"
```

Réaffichez la branche actuelle :

```
git branch
```

Affichez le contenu du dossier caché *.git* :

```
# windows
dir /s .git
```

```
# GNU/Linux
ls -R .git
tree .git
```

C'est ce dossier qui contient non seulement la configuration locale déjà faite (voir le fichier *config*), mais également toutes les révisions de code source présentes et à venir… Comprenez qu'il ne faut SURTOUT PAS l'effacer, sans quoi vous perdez définitivement tout votre projet !

N.B. : vous pouvez aussi créer un fichier caché *.gitignore* à la racine de votre projet, dans lequel chaque ligne désignera, en mode relatif, un fichier ou un dossier ignoré par l'index. On y met toujours en premier le *.gitignore* lui-même…

Imaginons ainsi qu'Alice veut se faire un dossier *sandbox* pour ses tests locaux, et qu'elle ne veut pas référencer ce dossier dans son GIT local.

```
mkdir sandbox
echo "allow from localhost" > sandbox/.htaccess
# windows
ipconfig
# GNU/Linux
ip a
# ou
hostname -I
echo "allow from $IP_ADDRESS" >> sandbox/.htaccess
echo "deny from All" >> sandbox/.htaccess
touch .gitignore
git status
echo ".gitignore" >> .gitignore
echo "sandbox" >> .gitignore
```

```
git status
```

Comme vous pouvez le constater, le fichier *.gitignore* ET le dossier *sandbox* sont bien ignorés...

E.2) Émulation d'un dépôt distant commun

Nous allons maintenant émuler le dépôt distant (généralement **gitlab**, **github** ou tout autre hébergeur, sauf qu'ici, nous n'aurons pas besoin de clés SSH pour s'authentifier sur le dépôt).

Allez dans le dossier qui va jouer le rôle de dépôt distant :

```
cd ../../le.depot.distant
```

Initialisez-le en tant que dépôt GIT distant :

```
git init . --bare
```

On peut vérifier qu'on est bien dans un dépôt de stockage et non un dépôt de travail :

```
git status
```

doit renvoyer un message d'erreur, et le contenu du dossier ressemble au contenu du *.git* d'un dossier de travail.

N.B. : **on ne peut pas avoir un dépôt qui soit à la fois un dépôt de stockage ET un dépôt de travail. C'est l'un ou l'autre !**

Dans la suite de ce document, *$PRJ* se référera au dossier de travail *CoursGit*.

Retournez dans le dossier de travail *$PRJ/alice/www.storcka.fr*.

```
cd ../alice/www.storcka.fr
pwd
```

On veut envoyer notre branche **master** sur l'espace de stockage distant. Pour cela, il faut d'abord donner un nom local au dépôt distant. En général, on le nomme **origin** pour original/originel.

```
git remote add origin $PRJ/le.depot.distant
# si vous avez fait une erreur :
# git remote remove origin
```

en replaçant *$PRJ* par **le chemin absolu** du dossier *le.depot.distant* !

Vérifiez la présence du nouveau dépôt distant :

```
git remote -v
```

Remarquez que le dépôt distant ne contient pas encore de branches :

```
git branch -r
```

Envoyez votre 1ère version du projet sur le dépôt distant :

```
git push origin master
```

Et revérifiez les branches distantes :

```
git branch -r
```

Demandez des renseignements sur la branche distante :

```
git remote show origin
```

N.B.: la branche **master** est généralement la branche publique officielle contenant la dernière version stable du projet.

E.3) Initialisation du site web

Bob est invité à participer au projet.

Dans un premier temps, il va devoir récupérer les fichiers d'Alice.

Allez dans *$PRJ/bob* :

```
cd $PRJ/bob
ls -l
```

Ici, comme il part de zéro, Bob choisit la méthode de clonage d'un dépôt distant. Dans tous les autres cas, on utilisera la méthode `git pull` que l'on verra plus loin, pour ramener les modifications des autres développeurs en local.

```
git clone $PRJ/le.depot.distant
ls -l
ls -l le.depot.distant
cat le.depot.distant/README.me
```

La dernière ligne permet de vérifier que le dossier contient bien le fichier *README.md* d'Alice.

Faites un :

```
cd le.depot.distant
git remote -v
```

et remarquez que le clonage a créé automatiquement le lien vers le dépôt distant **origin**, évitant à Bob de devoir le redéclarer de son côté...

Pour simplifier la suite, Bob décide de renommer son dossier *le.depot.distant* en *www.storcka.fr* :

Retournez dans *$PRJ/bob* :

```
cd $PRJ/bob
```

Renommez *le.depot.distant* en *www.storcka.fr* et allez dans le dossier :

```
mv le.depot.distant www.storcka.fr
cd www.storcka.fr
```

Configurez les identifiants GIT locaux de Bob :

```
git config user.name Bob
git config user.email bob@storcka.fr
git config color.ui auto
git config -l
```

Bob décide d'écrire le premier fichier du site web nommé *index.html* :

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width,
initial-scale=1">
 <title>D\'Storcka</title>
</head>
<body>
 <p>Site en construction...</p>
</body>
</html>
```

Indexez-le :

```
git add .
```

Vérifiez l'indexation :

```
git status
```

Envoyez les changements dans le dépôt GIT local :

```
git commit -m "Ajout du fichier index.html dans
master"
```

Vérifiez le changement :

```
git status
```

GIT indique que la branche **master** du dépôt local de Bob est en avance de 1 commit sur la branche **master** du dépôt distant.

Publiez les changements dans le dépôt GIT distant :

```
git push origin master
```

Vérifiez le dépôt distant :

```
git remote show origin
```

Vérifiez les révisions du projet :

```
git log
git log --oneline
```

E.4) Ajout authentification

Le dépôt master contient pour le moment le fichier *index.html* de Bob et le fichier *README.md* d'Alice.

Pendant que Bob était en train de coder le fichier *index.html*, Alice a décidé de s'attaquer au formulaire d'authentification principal.

Retournez sous *$PRJ/alice/www.storcka.fr* :

```
cd $PRJ/alice/www.storcka.fr
```

Remarquez bien que pour le moment, Alice n'a pas encore intégré les changements de Bob...

```
# windows
dir
# GNU/Linux
tree
```

Alice décide de créer une nouvelle branche **dev** qui sera la branche de développement principale pour tous les participants au projet, et une branche locale **feature/auth** pour développer sa fonctionnalité d'authentification.

Créez les deux branches :

```
git branch dev
git branch feature/auth
```

Affichez la branche actuelle (normalement **master**...) :

```
git branch
```

Changez vers la branche **feature/auth** :

```
git switch feature/auth
```

N.B. : les nouvelles versions de GIT ont introduit l'option **switch** qui fait la même chose que l'option historique **checkout**, mais est peut-être plus parlante pour les débutants... À vous de voir celle que vous préférez !

Vérifiez le changement :

```
git branch
```

Remarquez que la branche **feature/auth** n'existe pas sur le dépôt distant :

```
git branch -r
```

Remarquez également que le **status** vous indique toujours la branche courante :

```
git status
```

Créez le sous-dossier **core** qui contiendra les fichiers PHP principaux du site web :

```
mkdir core
```

Créez le fichier *core/const.inc.php* qui fixe les constantes PHP, les dossiers où trouver les classes, les encodages de caractères.... :

```
<?php
// for strtoupper and other string functions
```

```
setlocale(LC_ALL,'fr_FR.UTF8');
setlocale(LC_TIME,'fr_FR.UTF8');
setlocale(LC_CTYPE,'fr_FR.UTF8');
date_default_timezone_set('Europe/Paris');
// session
session_name('storcka');
session_start();
// for CLI debugging
define('ISCLI', PHP_SAPI === 'cli');
if (ISCLI) {
 parse_str(implode('&', array_slice($argv, 1)),
$_GET);
 $args = $_GET;
} else {
 $args = $_POST;
}
// errors
error_reporting(E_ALL ^ E_NOTICE);
// autoload classes
function autoload1 ($classname) {
 $target =
__DIR__.'/'.strtolower($classname).'.class.php';
 if (file_exists($target)) {
   include_once($target);
 }
}
spl_autoload_register('autoload1');
// authentication on startup
Auth::onstartup();
```

Ajoutez le fichier à l'index et faites une révision :

```
git status
```

```
git add .
git status
git commit -m "Ajout de const.inc.php"
git status
git log --oneline
```

Créez le fichier *core/page.class.php* :

```
<?php
class Page {
 public static function header () {
   header('Content-Type: text/plain; charset=UTF-
8');
   print '<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width,
initial-scale=1">
 <title>D\'Storcka</title>
</head>
<body>';
   if (isset($_SESSION['auth/id']) &&
mb_strlen($_SESSION['auth/id'])) {
     print Auth::banner();
   }
 }
 public static function footer () {
   print '
</body>
</html>';
 }
}
```

Ajoutez le fichier à l'index et faites une révision :

```
git status
git add .
git status
git commit -m "Ajout de page.class.php"
git status
git log --oneline
```

Créez le fichier *core/auth.class.php* :

```
<?php
class Auth {
 public static function form () {
   return '
   <div class="auth_popup">
     <form action="/" method="post"
enctype="multipart/form-data">
       <label for="login">Identifiant</label> :
<input type="text" name="login" id="login"
value=""/><br/>
       <label for="pass">Mot de passe</label> :
<input type="password" name="pass" id="pass"
value=""/><br/>
       <button type="submit">Envoyer</button>
     </form>
   </div>';
 }
 public static function onstartup () {
   global $args;
   if (isset($args['login']) &&
isset($args['pass'])) {
     $_SESSION['auth/id'] = '';
     $_SESSION['auth/txt'] = '';
     if (!strcmp($args['login'],'james') && !
strcmp($args['pass'],'bond')) {
       $_SESSION['auth/id'] = '007';
       $_SESSION['auth/txt'] = 'James Bond';
     }
   }
 }
 public static function banner () {
   return '
   <div
class="auth_bar">'.htmlentities($_SESSION['auth/txt'
]).' <button type="button" id="disconnect">Se
déconnecter</button></div>
   ';
 }
 public static function onform () {
   global $args;
   if (!isset($_SESSION['auth/id']) || !
mb_strlen($_SESSION['auth/id'])) {
     if (isset($args['login']) ||
isset($args['pass'])) {
       print '<p class="error">Identifiants erronés
!</p>';
     }
     print Auth::form();
   } else {
     if (!ISCLI) {
       print '<p class="info">Bienvenue !</p>';
     }
   }
 }
```

```
}
```

Ajoutez le fichier à l'index et faites une révision :

```
git status
git add .
git status
git commit -m "Ajout de auth.class.php"
git status
git log --oneline
```

Créez enfin le fichier *index.php* :

```
<?php
include('core/const.inc.php');
Page::header();
Auth::onform();
Page::footer();
```

Ajoutez le fichier à l'index et faites une révision :

```
git status
git add .
git status
git commit -m "Ajout de index.php"
git status
git log --oneline
```

Et testez enfin le script PHP en ligne de commande

```
php index.php
```

Comme vous pouvez le constater, sans arguments en ligne de commande (**CLI** pour Command Line Interface), notre script *index.php* renvoie un formulaire HTML vide d'authentification avec login/pass et bouton de validation.

Dans *const.inc.php*, nous avons utilisé une astuce qui permet de transmettre des arguments au script sous la forme **clé=valeur**, ce qui est très pratique pour faire des tests unitaires dans les gros projets...

C'est ce que nous allons ici utiliser pour tester notre formulaire.

```
php index.php login=james pass=bond
```

Si PHP râle ici en disant qu'il ne trouve pas la fonction `mb_strlen()`, c'est simplement qu'on a oublié d'installer la librairie PHP optionnelle contenant cette fonction.

Sous Debian, un simple :

```
apt install php7.4-mbstring  php-mbstring
```

permet de rajouter cette librairie essentielle capable de traiter les chaînes en UTF8 (multibyte strings). On rappelle en effet qu'en informatique, **les caractères UTF8 qui sont aujourd'hui la norme en conception, et occupent entre 1 et 4 octets suivant les symboles**… On ne peut donc se contenter de fonctions qui comptent le nombre d'octects, sous risque de se tromper lourdement !

N.B.: en CLI, PHP ne reconnaît pas les sessions ! Quand on veut émuler une page d'administration ou une page d'espace privé, où l'on doit rester authentifier, il faut donc ruser...

Pour revenir à notre GIT, Alice vient donc de créer *core/const.inc.php*, *core/page.class.php*, *core/auth.class.php* et *index.php*, en faisant à chaque fois une révision à part (dans la pratique, une révision pour l'ensemble aurait été suffisante…).

En attendant, Alice est contente. Sa fonctionnalité semble fonctionner. Elle décide de fusionner sa fonctionnalité à la branche **dev**.

Problème : Alice ne sait plus si la branche **dev** contient déjà du code. Elle choisit de le vérifier et se déplace donc sur la branche.

Allez sur la branche **dev** :

```
git switch dev
```

Affichez le contenu de la branche :

```
dir / ls
```

Et là : c'est le drame : il n'y a que le *README.md* et le bac à sable ! Tous les fichiers de la fonctionnalité **feature/auth**, que nous venions juste de créer, ont disparu !

Pas de panique : en fait, ils sont toujours présents, mais dans la branche d'origine **feature/auth** !

On va retourner sur la branche pour le vérifier :

```
git switch feature/auth
ls -lR
```

Et là vous comprenez qu'avec GIT, votre dossier de travail devient 100% virtuel : il est systématiquement remis à jour avec la branche sur laquelle vous travaillez, tout en conservant les modifications des autres branches !

Voyant qu'il n'y aura pas de conflits potentiels avec d'autres fichiers, Alice choisit de fusionner sa branche **feature/auth** avec la branche **dev**.

N.B.: la fusion se fait toujours en se déplaçant d'abord sur la branche de destination.

Retournez donc sur la branche **dev** :

```
git switch dev
```

Fusionnez la branche **feature/auth** en mode **fast forward** :

```
git log --oneline --graph
git merge feature/auth
git log --oneline --graph
```

GIT vous indique qu'il a fait la fusion en mode **Fast-Forward** (avance rapide). Cela signifie qu'il n'y a pas eu de révisions sur la branche **dev** depuis la création de la branche **feature/auth**.

Comme il n'y a pas alors de risques de conflits, GIT a directement repris toutes les révisions de la branche **feature/auth** pour les rajouter à la suite de la branche **dev**.

C'est le mode de fusion le plus simple.

Mais côté historique, il a un inconvénient majeur : nous conservons toutes les informations des 4 révisions qui ont eu lieu dans la branche **feature/auth**. Or à part Alice, les autres développeurs n'en ont cure…

Imaginez un projet où chaque développeur cumule des dizaines de révisions dans sa branche **feature**, puis les copie dans la branche commune : on risque de vite saturer l'historique en informations inutiles, et d'agacer les autres participants !

Tout dépend ensuite de la stratégie de votre groupe de travail. Certains préfèrent tout conserver. D'autres préfèrent optimiser.

Dans notre cas, Alice décide que les commentaires de sa branche **feature/auth** ne sont pas indispensables au suivi du projet. Elle chosit donc corriger sa fusion précédente.

Vérifiez bien que vous êtes toujours sur la branche **dev** :

```
git branch
ls / dir
```

Effacez les 4 dernièrs niveaux de révisions :

```
git log --oneline --graph
git reset --hard HEAD~4
git log --oneline --graph
ls / dir
```

Refaites la fusion de **feature/auth** dans **dev** avec l'option *--no-ff* :

```
git log --oneline --graph
git merge feature/auth --no-ff
```

Modifiez le commentaire en :

```
Fusion de feature/auth dans dev
```

Réaffichez le journal :

```
git log --oneline --graph
```

Comme vous pouvez le constater, cette option a forcé la création d'un révision sur la branche **dev**, ignorant le mode Fast-Forward par défaut.

Cela étant, cette option a laissé toutes les informations de révisions que nous voulions précisément éviter !

Nous allons donc revenir en arrière d'une révision sur notre branche **dev** :

```
git branch
git reset --hard HEAD~
git log --oneline --graph
```

Et cette fois, nous allons utiliser l'option `--squash` qui va rajouter nos nouveaux fichiers et dossiers dans notre branche **dev**, en mettant à jour l'index, MAIS sans créer de nouvelle révision :

```
git merge feature/auth --squash
git status
git log --oneline --graph
```

Vérifiez la présence des nouveaux fichiers dans **dev** :

```
git branch
ls -lR / dir
```

Alice peut maintenant créer une révision pour finaliser l'intégration de sa branche :

```
git commit -m "Ajout de la branche feature/auth dans
dev"
```

Remarquez que la branche **feature/auth** existe toujours.

Retournez-y :

```
git switch feature/auth
ls -lR
```

Alice ne veut pas surcharger inutilement le projet. Maintenant que la branche **dev** contient ses ajouts, elle choisit d'effacer l'ancienne branche devenue inutile.

Retournez sur la branche **dev** :

```
git switch dev
```

Effacez la branche **feature/auth** :

```
git log --oneline --graph
git branch -d feature/auth
```

GIT prévient que la branche n'est pas totalement fusionnée et refuse de l'effacer. Nous choisissons ici d'outrepasser ce message d'erreur car nous n'avons pas fait d'autres modifications depuis la dernière révision, où nous avions intégrer l'ensemble des modifications.

```
git branch -D feature/auth
```

Vérification du journal :

```
git log --graph
git log --graph --oneline
```

Alice choisit enfin de publier les changements de sa branche dev sur le site distant :

```
git push origin dev
```

E.5) Routage des URI

La réunion d'équipe fait apparaître le besoin d'un routage des URI, utilisant des espaces de nom, des classes et fonctions.

Alice est chargée de développer le module. Bob est chargé de développer un module **contact** en respectant simplement la convention de création des modules mise au point pendant la réunion.

Alice retourne donc à son bureau, et commence à créer une branche **feature/route**.

Créez la nouvelle branche :

```
git branch feature/route
```

Placez-vous dessus :

```
git switch feature/route
```

Modifiez le *index.php* :

```
<?php
include('core/const.inc.php');
Page::header();
try {
 print $_URI->handle();
} catch (Exception $e) {
 print '
 <p>Désolé mais une erreur est survenue. L\'équipe a été prévenue. Merci de revenir plus tard.</p>';
 #mail('webmaster@storcka.net','Erreur sur le site', $e->getMessage());
```

```
}
Page::footer();
```

Créez le fichier *phpinfo.php* :

```
<?php
phpinfo();
```

Modifiez le *core/auth.class.php* :

```
<?php
class Auth {
 public function __construct () {
 }
 public static function form () {
   return '
   <div class="auth_popup">
     <form action="/" method="post"
enctype="multipart/form-data">
       <label for="login">Identifiant</label> :
<input type="text" name="login" id="login"
value=""/><br/>
       <label for="pass">Mot de passe</label> :
<input type="password" name="pass" id="pass"
value=""/><br/>
       <button type="submit">Envoyer</button>
     </form>
   </div>';
 }
 public function onstartup () {
   global $args;
   if (isset($args['login']) &&
isset($args['pass'])) {
     $_SESSION['auth/id'] = '';
     $_SESSION['auth/txt'] = '';
     if (!strcmp($args['login'],'james') && !
strcmp($args['pass'],'bond')) {
       $_SESSION['auth/id'] = '007';
       $_SESSION['auth/txt'] = 'James Bond';
     }
   }
 }
 public static function banner () {
   return '
   <div
class="auth_bar">'.htmlentities($_SESSION['auth/txt'
]).' <button type="button" id="disconnect">Se
déconnecter</button></div>
   ';
 }
 public static function onform () {
   global $args;
   if (!isset($_SESSION['auth/id']) || !
mb_strlen($_SESSION['auth/id'])) {
     if (isset($args['login']) ||
isset($args['pass'])) {
       print '<p class="error">Identifiants erronés
!</p>';
     }
     print Auth::form();
   } else {
     if (!ISCLI) {
       print '<p class="info">Bienvenue !</p>';
     }
   }
```

```
 }
}
```

Modifiez le *core/const.inc.php* :

```
<?php
// for strtoupper and other string functions
setlocale(LC_ALL,'fr_FR.UTF8');
setlocale(LC_TIME,'fr_FR.UTF8');
setlocale(LC_CTYPE,'fr_FR.UTF8');
date_default_timezone_set('Europe/Paris');
// session
session_name('STORCKA');
session_start();
// for CLI debugging
define('ISCLI', PHP_SAPI === 'cli');
if (ISCLI) {
 parse_str(implode('&', array_slice($argv, 1)),
$_GET);
 $args = $_GET;
} else {
 $args = $_POST;
}
// errors
error_reporting(E_ALL ^ E_NOTICE);
// autoload classes
function autoload1 ($classname) {
 $target =
__DIR__.'/'.strtolower($classname).'.class.php';
 if (file_exists($target)) {
   include_once($target);
 }
}
```

```
spl_autoload_register('autoload1');
// Add route mechanismus
global $_URI;
$_URI = new URI();
// Add modules
foreach (glob('mod/*/module.inc.php') as $key =>
$val) {
 include_once($val);
}
// authentication on startup
global $_Auth;
$_Auth  = new Auth();
$_Auth->onstartup();
```

Modifiez le *core/page.class.php* :

```
<?php
class Page {
 public static function header () {
   global $_Auth;
   header('Content-Type: text/plain; charset=UTF-
8');
   print '<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width,
initial-scale=1">
 <title>D\'Storcka</title>
</head>
<body>
';
```

```
    if (isset($_SESSION['auth/id']) &&
mb_strlen($_SESSION['auth/id'])) {
      print $_Auth->banner();
    }
  }
  public static function footer () {
    print '</body>
</html>';
  }
}
```

Ajoutez le fichier *core/uri.class.php* :

```
<?php
class URI {
  public function __construct () {
    $this->uri = [];
  }
  public function register ($path, $namespace,
$class, $func) {
    if (!isset($this->uri[$path])) {
      $this->uri[$path] = [
        'namespace' => $namespace,
        'class' => $class,
        'func' => $func,
      ];
    }
  }
  public function call ($path) {
    if (!isset($this->uri[$path])) {
      throw new Exception('Path '.$path .' not found
!');
    }
    $uri = $this->uri[$path];
    $n = $uri['namespace'];
    $c = $uri['class'];
    $f = $uri['func'];
    $nc = "$n\\$c";
    return (new $nc)->$f();
  }
  public function handle () {
    global $args;
    if (!isset($args['path'])) {
      return $this->call('/');
    }
    return $this->call($args['path']);
  }
}
```

Retournez sous *$PRJ/alice/www.storcka.fr* :

```
cd $PRJ/alice/www.storcka.fr
```

Créez le dossier des modules :

```
mkdir mod
```

Créez le sous-dossier du contributeur *fr.storcka* dans les modules :

```
mkdir mod/fr.storcka/
```

Créez le sous-dossier page dans les modules de *storcka.fr* :

```
mkdir mod/fr.storcka/page
```

Créez le fichier *mod/fr.storcka/module.inc.php* :

```
<?php
include_once(__DIR__.'/page.class.php');
global $_URI;
$_URI->register('/','fr\storcka','Page','handle');
$_URI->register('/index.html','fr\
storcka','Page','handle');
$_URI->register('/index.php','fr\
storcka','Page','handle');
```

Créez le fichier *mod/fr.storcka/page.class.php* :

```
<?php
namespace fr\storcka;
class Page {
 public function __construct () {
 }
 public function php ($pagename) {
   $file = __DIR__.'/page/'.$pagename.'.php';
   if (file_exists($file)) {
     ob_start();
     include_once($file);
     $res = ob_get_contents();
     ob_end_clean();
     return $res;
   }
 }
 public function html ($pagename) {
   $file = __DIR__.'/page/'.$pagename.'.html';
   if (file_exists($file)) {
     return file_get_contents($file);
   }
 }
 public function call ($pagename) {
   //print "calling $pagename...\n";
   $output = '';
   $output .= $this->php($pagename);
   $output .= $this->html($pagename);
   return $output;
 }
 public function handle () {
   global $args;
   if (!isset($args['page'])) {
     return $this->call('homepage');
   }
   return $this->call($args['page']);
 }
}
```

Créez le fichier *mod/fr.storcka/page/homepage.html* :

```
<p>Bienvenue chez les Cigognes !</p>
```

Retournez sous *$PRJ/alice/www.storcka.fr* :

```
cd $PRJ/alice/www.storcka.fr
```

Testez le script :

```
php index.php
php index.php login=james pass=bond
```

Testez également les informations serveur de PHP :

```
php phpinfo.php|less
```

Et vérifiez le contenu :

```
tree
.
├── core
│   ├── auth.class.php
│   ├── const.inc.php
│   ├── page.class.php
│   └── uri.class.php
├── index.php
├── mod
│   └── fr.storcka
│       ├── module.inc.php
│       ├── page
│       │   └── homepage.html
│       └── page.class.php
├── phpinfo.php
├── README.md
└── sandbox
```

Alice est satisfaite : son script fonctionne correctement. Elle décide donc de publier ses changements dans son GIT local.

D'abord elle vérifie l'état de l'index local :

```
git status
```

Elle y rajoute l'ensemble des fichiers (créés + modifiés) :

```
git add .
```

Alice revérifie ensuite l'état de l'index local et la branche actuelle (**feature/route**) :

```
git status
```

Tout est ok : elle publie donc la nouvelle version en local :

```
git commit -m "Ajout du routage et de la gestion des
modules + premier module Page"
git status
git log --graph --oneline
```

Mais là, elle se souvient soudainement qu'on ne laisse JAMAIS traîner, sur un site web, un fichier donnant des informations sur le serveur PHP - qui plus est à la racine…

Elle déplace donc son *phpinfo.php* dans son dossier *sandbox*.

```
mv phpinfo.php sandbox
```

Et refait tout de suite une révision de correction :

```
git add -A
git status
git commit -m "Déplacement de phpinfo vers sandbox"
```

Petit rappel :

➔ `git add .` : ajoute les fichiers nouveaux/modifiés à l'index, mais ignore les fichiers effacés dans le dossier de travail.

➔ `git add -A` (`--all`) : comme `git add .`, mais en plus, tient compte des fichiers effacés.

Merci de lire cette petite aide en ligne, pour revoir les autres commandes...

Rappel : Alice est toujours actuellement sur la branche **feature/route**.

Par curiosité, elle veut revoir l'ancienne version du fichier *index.php*.

Elle commence par afficher le journal des versions :

```
git log
```

Elle repère les numéros de révision (**commit**), sous la forme d'une longue chaîne de caractères, la dernière version étant celle indiquée par le pointeur *HEAD*.

Mais devant la longueur de ce dernier, elle décide rapidement de restreindre sa recherche au seul fichier :

```
git log -- index.php
```

Elle voit deux révisions qui ont aboutit à la création du fichier.

Elle repère le numéro de la version la plus ancienne du code (juste après le mot-clé `commit`) que nous nommerons `$FIRST_COMMIT_HASH` dans la suite.

Afficher le contenu actuel et la version courante d'*index.php* :

```
# windows
dir /s
# GNU/Linux
tree
cat index.php
```

Retournez à la version précédente en remplaçant `$FIRST_COMMIT_HASH` par le numéro de version préalablement repéré :

```
git checkout $FIRST_COMMIT_HASH
```

Affichez les fichiers qui étaient présents dans cette révision :

```
# windows
dir /s
# GNU/Linux
tree
```

Afficher le contenu du *index.php* de l'époque :

```
# windows
type index.php
# GNU/Linux
cat index.php
```

N.B. : vous êtes en pointeur **HEAD détaché**, ce qui signifie que vous êtes sur une branche virtuelle du projet, et par conséquent, si vous faites des changements dans les fichiers, GIT ne saura pas quoi en faire par défaut !

```
git branch
git status
```

De là, deux solutions :

➔ soit vous faites réellement des modifications avec un **commit** à chaque fois, et vous les attribuez au final à une nouvelle branche, que vous pouvez créer avec le raccourci `git switch -c nom_de_la_nouvelle_branche`.

➔ soit vous abandonnez vos modifications sans possibilité de retour...

Comprenez qu'en règle générale, quand on revient en arrière avec `git checkout`, c'est surtout pour revoir d'anciens fichiers, sans les

modifier, ou créer une nouvelle branche à partir d'une ancienne révision...

Alice veut maintenant revenir à la branche en cours.

Retournez sur la branche **feature/route** avec :

```
git checkout feature/route
```

Constatez le retour des fichiers et dossiers :

```
# windows
dir /s
# GNU/Linux
tree
cat index.php
```

Notons que la manière de faire d'Alice n'est pas ici la plus futée pour comparer des fichiers textes ou des arborescences de fichiers !

En effet : sous GNU/Linux, on trouve des utilitaires console (`diff`, `vimdiff`) ou graphique (`meld`) bien plus efficaces pour comparer des fichiers textes, mais aussi des dossiers !

GIT a bien entendu repris cette logique, et intégré ces utilitaires indispensables aux développeurs...

Alice décide ainsi d'afficher les différences opérées sur *index.php*, mais cette fois au format diff :

```
git log
git diff $FIRST_COMMIT_HASH -- index.php
```

Les lignes **+** (en vert) représentent les ajouts.

Les lignes en **–** (en rouge) représentent les retraits.

Alice décide ensuite de fusionner sa branche avec la branche **dev** :

```
git branch
git switch dev
git merge feature/route
```

Remarquez que la fusion ne vous a pas demandé de rajouter un commentaire, et pour cause : elle a été faite en mode avance rapide (Fast Forward), puisque rien n'avait modifié sur la branche **dev**, depuis la création de la branche **feature/route**.

Alice choisit ici de ne pas effacer la branche **feature/route** pour le moment.

Remarquez bien qu'à cet instant, Alice n'a pas encore envoyé ses modifications de la branche **dev** sur le dépôt distant… Elle n'a fait que fusionner la fonctionnalité à sa branche **dev** locale.

E.6) Le formulaire de contact

Des réunions de travail, Bob et Roger savent qu'une branche **dev** existe à distance, mais pour le moment, ils ne l'ont pas encore utilisée.

Bob a reçu la mission de s'occuper du formulaire de contact.

Retournez donc sous *$PRJ/bob/www.storcka.fr* :

```
cd $PRJ/bob/www.storcka.fr
```

Remarquez que Bob n'a pour le moment que la branche master du projet en local :

```
git remote show origin
git branch
git branch -r
# windows
dir /s
# GNU/Linux
tree
```

Ramenez la branche distante **dev** en local :

```
git fetch origin dev
git branch
```

N.B.:

➔ `git fetch` télécharge les dernières révisions d'une branche donnée depuis un dépôt distant, mais ne modifie pas votre dépôt local. L'idée de `git fetch` est juste de disposer des fichiers nécessaires au travail, en conservant l'état actuel du dépôt local.

➔ `git pull` est l'abréviation de `git fetch` + `git merge`. La commande fait donc la même chose que l'option fetch, mais en plus, les fichiers distants sont fusionnés avec les fichiers locaux, en suivant le règlement des conflits. Suivant les changements opérés par les uns et les autres, l'opération peut donc s'avérer complexe et longue.

Déplacez-vous sur la branche **dev** :

```
git switch dev
```

Vérifiez les fichiers présents :

```
# windows
dir /s
# GNU/Linux
tree
```

Un peu fatigué, et parce qu'il avait mal compris les instructions pendant la réunion, Bob commence à recoder le fichier *index.php* du projet dans sa branche **dev**.

Modifiez donc le *index.php* :

```
<?php
include('core/const.inc.php');
Page::header();
switch ($args['route']) {
 case 'contact':
   Contact::handle();
   break;
 case 'auth':
   Auth::handle();
   break;
 default:
   Page::welcome();
}
Page::footer();
```

Il rajoute son fichier et créé une révision :

```
git add .
git commit -m "Modification de index.php"
```

Et là, il se rend compte de son erreur… il affiche alors son journal des révisions :

```
git log
```

Vu qu'il ne s'est pas encore attaqué au formulaire de contact proprement dit, il se dit qu'il vaudrait mieux revenir rapidement à la version précédente, en oubliant ses propres modifications. Plusieurs syntaxes GIT sont alors possibles. Il choisit la syntaxe

```
cat index.php
git reset --hard HEAD^
cat index.php
```

L'option **--hard** est très importante : elle dit à GIT d'oublier définitivement toutes les modifications en cours, qui seront donc définitivement perdues !

Vérifiez le journal modifié :

```
git log
```

Bob s'attaque maintenant à la création du formulaire de contact.

Il commence par créer une branche pour sa fonctionnalité à partir de la branche **dev** où il se trouve.

Créez la branche **feature/contact** :

```
git branch feature/contact
```

Allez sur la nouvelle branche :

```
git switch feature/contact
```

Créez le dossier *mod/fr.storcka* :

```
mkdir -p mod/fr.storcka
```

Résumé : Bob a ramené la branche dev distante, alors qu'Alice 'a pas encore versé ses changements dessus, et il a fait une petite erreur qu'il a corrigé en local en revenant à la révision précédente, donc en perdant ses modifications via l'option `--hard`.

Pendant ce temps-là, Alice a décidé de publier ses modifications de la branche **dev** sur le dépôt **origin**.

Retournez donc sous *$PRJ/alice/www.storcka.fr* :

```
cd $PRJ/alice/www.storcka.fr
```

Allez sur la branche **dev** :

```
git switch dev
```

Publier les modifications dans le dépôt **origin** :

```
git push origin dev
```

Les branches **dev** sont maintenant différentes : celles d'Alice et du dépôt **origin** sont à jour. Celle de Bob est encore l'ancienne version.

Bob commence à coder son formulaire de contact.

Retournez donc sous *$PRJ/bob/www.storcka.fr* :

```
cd $PRJ/bob/www.storcka.fr
```

Retournez sur la branche **feature/contact** :

```
git switch feature/contact
```

Créez le fichier *mod/fr.storcka/module.inc.php* :

```
<?php
include_once(__DIR__.'/contact.class.php');
global $_URI;
$_URI->register('/contact','fr\
storcka','Contact','form');
$_URI->register('/contact/post','fr\
storcka','Contact','handle');
```

Créez le fichier *mod/fr.storcka/contact.class.php* :

```
<?php
namespace fr\storcka;
class Contact {
 public function __construct () {
   $this->fields = [
     'fn' => [
       'label' => 'Prénom',
       'val' => '',
     ],
     'ln' => [
       'label' => 'Nom',
       'val' => '',
     ],
     'email' => [
       'label' => 'Courriel',
       'val' => '',
     ],
     'mess' => [
       'label' => 'Message',
       'val' => '',
     ],
   ];
 }
 public function form () {
   $output = '';
   reset($this->fields);
   foreach ($this->fields as $key => $val) {
     $output .= '
     <div class="form_field">
       <label
for="'.$key.'">'.$val['label'].'</label> : <input
type="text" name="'.$key.'" id="'.$key.'"
value="'.htmlentities($val['val']).'" />
     </div>';
   }
   return '
   <form action="/contact/post" method="post"
enctype="multipart/form-data">'.$output.'
     <div class="form_ctrl">
       <button type="submit">Envoyer</button>
     </div>
   </form>';
 }
 public function err ($mess) {
   return '
   <p class="error">'.$mess.'</p>';
 }
 public function handle () {
   global $args;
   $output = '';
   reset($this->fields);
   foreach ($this->fields as $key => $val) {
     $out = '';
     $data = trim($args[$key]);
     if (!mb_strlen($data)) {
```

```
        $out .= $this->err('Le champ "'.
$val['label'].'" est vide !');
      }
      if (!strcmp('email',$key) && mb_strlen($data))
{
        if (!filter_var($data,
FILTER_VALIDATE_EMAIL)) {
          $out .= $this->err('Le champ "'.
$val['label'].'" est erroné !');
        }
      }
      if (!mb_strlen($out)) {
        $this->fields[$key]['val'] = $data;
      }
      $output .= $out;
    }
    if (mb_strlen($output)) { // some errors
occured...
      $output .= $this->form();
    } else {
      // record datas somewhere...
      // mail('contact@storcka.fr','Contact
site',"$fn,$ln,$email,$mess");
      $output = '
      <p>Formulaire envoyé ! Merci pour votre
message !</p>';
    }
    return $output;
  }
}
```

Bob ayant suivi le schéma de création des modules vu en réunion avec ses collègues, mais ne pouvant tester complètement le code, il vérifie juste s'il n'y a pas d'erreurs de syntaxe dans la classe PHP.

Testez le script :

```
php mod/fr.storcka/contact.class.php
```

Pas de message d'erreur retourné = pas d'erreur PHP ! Tout est ok !

Toujours dans *$PRJ/bob/www.storcka.f*r :

```
cd $PRJ/bob/www.storcka.fr
```

Vérifiez que vous êtes bien sur la branche **feature/contact** :

```
git status
```

Ajoutez les fichiers à l'index :

```
git add .
```

Revérifiez le statut :

```
git status
```

Créez la révision :

```
git commit -m "Ajout du formulaire de contact"
```

La branche de Bob affiche les fichiers suivants à cet instant :

```
tree
.
├── core
│   ├── auth.class.php
```

```
│   ├── const.inc.php
│   └── page.class.php
├── index.php
├── mod
│   └── fr.storcka
│       ├── contact.class.php
│       └── module.inc.php
└── README.md
```

Bob veut fusionner la branche **feature/contact** avec la branche **dev**.

Allez sur la branche de destination :

```
git switch dev
```

Fusionnez la branche :

```
git merge feature/contact
```

Bob tente alors d'envoyer sa branche **dev** sur le dépôt distant **origin** :

```
git push origin dev
```

Et là : c'est le drame !!!

Les changements téléversés par Alice avant Bob font que la branche **dev** locale de Bob n'est plus à jour, et il y a des conflits !

Si on rentre un peu plus dans les détails, on voit par exemple le conflit sur le fichier *module.inc.php* créé par Alice, et envoyé sur la branche **dev** du dépôt **origin**

Affichez notre dépôt **dev** local en mode récursif :

```
git ls-tree -r dev
```

Affichez le cache local de notre dépôt **dev** distant, en mode récursif :

```
git ls-tree -r remotes/origin/dev
```

La ligne suivante renvoie une erreur :

```
git show
remotes/origin/dev:mod/fr.storcka/module.inc.php
```

Cette erreur est normale, et vient nous dire que pour le moment, le fichier *mod/fr.storcka/module.inc.php* n'a pas été trouvé dans notre cache local du dépôt distant **dev**.

Remettez donc à jour le cache local de notre dépôt **dev** distant :

```
git fetch origin dev
```

Remarquez d'abord que notre dépôt **dev** local n'a pas évolué (on n'a pas retravaillé dessus) :

```
git ls-tree -r dev
```

Nos fichiers locaux ont été conservés.

En revanche, le cache local du dépôt **dev** distant ayant été remis à jour, on s'aperçoit que la liste des nouveaux fichiers a évolué. **MAIS SANS ÉCRASER NOS FICHIERS LOCAUX !**

```
git ls-tree -r remotes/origin/dev
```

C'est ce qui distingue le `fetch` du `pull`. Le `fetch` remet à jour la liste des fichiers/dossiers présents sur le dépôt distant dans le cache local, sans affecter les fichiers locaux. Le *pull* en revanche

aurait non seulement fait cette opération, mais aurait en plus provoqué la fusion immédiate de tous les fichiers distants avec notre branche **dev** locale !

On remarque ainsi que le dépôt distant contient déjà un fichier *mod/fr.storcka/module.inc.php* qui va rentrer en conflit avec notre version locale...

```
git show
remotes/origin/dev:mod/fr.storcka/module.inc.php
cat mod/fr.storcka/module.inc.php
```

Vous pouvez refaire la commande d'envoi des modifications locales sur **dev** à distance pour revoir le message d'erreur renvoyé par GIT :

```
git push origin dev
```

Bob n'a donc pas le choix ici : il doit d'abord fusionner les fichiers créés par les autres développeurs depuis **origin/dev** dans son dépôt local.

```
git status
git log --oneline --graph
ls -lR
git pull origin dev
```

Rappel : `git pull` = `git fetch` + `git merge` !

Git signale un problème qui empêche le `git merge` : Bob n'a pas encore spécifié qu'elle méthode de fusion il souhaite utiliser par défaut.

Bob choisit donc la méthode de fusion par défaut :

```
git config pull.rebase false
```

Il retente ensuite l'intégration :

```
git pull origin dev
git status
git log --oneline --graph
ls -lR
```

Cette fois,la commande lui indique clairement le conflit : dans notre cas, le fichier *mod/fr.storcka/module.inc.php*.

Éditez donc le fichier *mod/fr.storcka/module.inc.php* :

```
notepad mod/fr.storcka/module.inc.php
nano mod/fr.storcka/module.inc.php
vi mod/fr.storcka/module.inc.php
```

Comme vous pouvez le constater, GIT a simplement fusionné les deux codes sources en conflit. C'est donc au développeur de faire le tri et de régler les problèmes ! (oui c'est honteux : GIT ne fait toujours pas le café…).

Réarrangez le fichier *mod/fr.storcka/module.inc.ph*p comme suit :

```
<?php
global $_URI;
include_once(__DIR__.'/page.class.php');
$_URI->register('/','fr\storcka','Page','handle');
$_URI->register('/index.html','fr\
storcka','Page','handle');
$_URI->register('/index.php','fr\
storcka','Page','handle');
include_once(__DIR__.'/contact.class.php');
```

```
$_URI->register('/contact','fr\
storcka','Contact','form');
$_URI->register('/contact/post','fr\
storcka','Contact','handle');
```

Bob veut en profiter pour tester son module de contact :

```
php index.php path='/contact'
php index.php path='/contact/post' fn='toto'
email='toto'
php index.php path='/contact/post' fn='toto'
ln='toto' email='toto@toto.fr' mess='coucou'
```

Tout est ok : Bob veut renvoyer la nouvelle branche modifiée sur le dépôt distant. !

Affichez le statut de l'index et vérifiez que vous êtes bien positionné sur la branche **dev** :

```
git status
```

Ajoutez les fichiers modifiés

```
git add -A
```

Réaffichez le statut de l'index :

```
git status
```

Créez la nouvelle version réglant le conflit :

```
git commit -m "Fusion du module de contact à la
branche dev"
```

Envoyez la nouvelle version sur le dépôt :

```
git push origin dev
```

Et regardez le journal principal :

```
git log --oneline --graph
```

Au final, les fichiers de Bob dans la branche dev sont donc les suivants :

```
tree
.
├── core
│   ├── auth.class.php
│   ├── const.inc.php
│   ├── page.class.php
│   └── uri.class.php
├── index.php
├── mod
│   └── fr.storcka
│       ├── contact.class.php
│       ├── module.inc.php
│       ├── page
│       │   └── homepage.html
│       └── page.class.php
└── README.md
```

E.7) Centralisation des erreurs

Le patron n'est pas satisfait : il aimerait que la gestion des erreurs soit centralisée pour tous les modules, mais aussi pour tous les scripts PHP du dossier *core/*.

Il charge Bob de faire les modifications, et de les intégrer à son module.

Créez la branche **feature/status** :

```
git branch feature/status
```

Basculez sur la nouvelle branche :

```
git switch feature/status
```

Créez le fichier *core/status.class.php* :

```
<?php
class Status {
 public function __construct () {
   $this->mess = [];
 }
 public function add ($type, $mess) {
   if (!isset($this->mess[$type])) {
     $this->mess[$type] = [];
   }
   $this->mess[$type][] = $mess;
 }
 public function error ($mess) {
   $this->add('error',$mess);
 }
 public function info ($mess) {
   $this->add('info',$mess);
 }
 public function warn ($mess) {
   $this->add('warn',$mess);
 }
}
```

Ajoutez le nouveau fichier à l'index local :

```
git add .
```

Validez la révision :

```
git commit -m "Ajout du status global"
```

Le lendemain en arrivant, Bob veut continuer ses modifications.

Modifiez le fichier *core/status.class.php* :

```
<?php
class Status {
 public function __construct () {
   $this->mess = [];
 }
 private function add ($type, $mess) {
   if (!isset($this->mess[$type])) {
     $this->mess[$type] = [];
   }
   $this->mess[$type][] = $mess;
 }
 public function err ($mess) {
   $this->add('error',$mess);
   return false;
 }
 public function info ($mess) {
   $this->add('info',$mess);
   return true;
 }
 public function warn ($mess) {
   $this->add('warn',$mess);
   return true;
```

```
 }
 public function html () {
   reset($this->mess);
   $output = '';
   foreach ($this->mess as $key => $val) {
     if (count($val)) {
       foreach ($val as $key2 => $val2) {
         $output .= '
     <p class="'.$key.'">'.$val2.'</p>';
       }
     }
   }
   return $output;
 }
}
```

Modifiez *core/const.inc.php* :

```
<?php
// for strtoupper and other string functions
setlocale(LC_ALL,'fr_FR.UTF8');
setlocale(LC_TIME,'fr_FR.UTF8');
setlocale(LC_CTYPE,'fr_FR.UTF8');
date_default_timezone_set('Europe/Paris');
// session
session_name('storcka');
session_start();
// for CLI debugging
define('ISCLI', PHP_SAPI === 'cli');
if (ISCLI) {
 parse_str(implode('&', array_slice($argv, 1)),
$_GET);
 $args = $_GET;
} else {
 $args = $_POST;
}
// errors
error_reporting(E_ALL ^ E_NOTICE);
// autoload classes
function autoload1 ($classname) {
 $target =
__DIR__.'/'.strtolower($classname).'.class.php';
 if (file_exists($target)) {
   include_once($target);
 }
}
spl_autoload_register('autoload1');
// Add status
$_Status = new Status();
// Add route mechanismus
global $_URI;
$_URI = new URI();
// Add modules
foreach (glob('mod/*/module.inc.php') as $key =>
$val) {
 include_once($val);
}
// authentication on startup
global $_Auth;
$_Auth  = new Auth();
$_Auth->onstartup();
```

Modifiez *core/page.class.php* :

```
<?php
class Page {
```

```
 public static function header () {
   global $_Auth, $_Status;
   header('Content-Type: text/plain; charset=UTF-
8');
   print '<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width,
initial-scale=1">
 <title>D\'Storcka</title>
</head>
<body>
';
   if (isset($_SESSION['auth/id']) &&
mb_strlen($_SESSION['auth/id'])) {
     print $_Auth->banner();
   }
   print $_Status->html();
 }
 public static function footer () {
   print '</body>
</html>';
 }
}
```

Modifiez *index.php* :

```
<?php
include('core/const.inc.php');
try {
 $output = $_URI->handle();
} catch (Exception $e) {
 $_Status->err('Désolé mais une erreur est survenue.
L\'équipe a été prévenue. Merci de revenir plus
tard.');
 $output = $_Status->html();
 #mail('webmaster@storcka.net','Erreur sur le site',
$e->getMessage());
}
Page::header();
print $output;
Page::footer();
```

Modifiez *mod/fr.storcka/contact.class.php* :

```
<?php
namespace fr\storcka;
class Contact {
 public function __construct () {
   $this->fields = [
     'fn' => [
       'label' => 'Prénom',
       'type' => 'text',
       'val' => '',
     ],
     'ln' => [
       'label' => 'Nom',
       'type' => 'text',
       'val' => '',
     ],
     'email' => [
       'label' => 'Courriel',
       'type' => 'text',
       'val' => '',
     ],
```

```
      'mess' => [
        'label' => 'Message',
        'type' => 'textarea',
        'val' => '',
      ],
    ];
  }
  public function form () {
    $output = '';
    reset($this->fields);
    foreach ($this->fields as $key => $val) {
      switch ($val['type']) {
        case 'text':
          $output .= '
      <div class="form_field">
        <label
for="'.$key.'">'.$val['label'].'</label> : <input
type="text" name="'.$key.'" id="'.$key.'"
value="'.htmlentities($val['val']).'" />
      </div>';
          break;
        case 'textarea':
          $output .= '
      <div class="form_field">
        <label
for="'.$key.'">'.$val['label'].'</label> : <textarea
name="'.$key.'" id="'.
$key.'">'.htmlentities($val['val']).'</textarea>
      </div>';
          break;
      }
    }
    return '
    <form action="/contact/post" method="post"
enctype="multipart/form-data">'.$output.'
      <div class="form_ctrl">
        <button type="submit">Envoyer</button>
      </div>
    </form>';
  }
  public function handle () {
    global $args, $_Status;
    $output = '';
    reset($this->fields);
    $ok1 = true;
    foreach ($this->fields as $key => $val) {
      $ok2 = true;
      $data = trim($args[$key]);
      if (!mb_strlen($data)) {
        $ok2 &= $_Status->err('Le champ "'.
$val['label'].'" est vide !');
      }
      if (!strcmp('email',$key) && mb_strlen($data))
{
        if (!filter_var($data,
FILTER_VALIDATE_EMAIL)) {
          $ok2 &= $_Status->err('Le champ "'.
$val['label'].'" est erroné !');
        }
      }
      if ($ok2) { // no errors
        $this->fields[$key]['val'] = $data;
      }
      $ok1 &= $ok2;
```

```
    }
    if (!$ok1) { // some errors occured...
      $output = $this->form();
    } else {
      // record datas somewhere...
      // mail('contact@storcka.fr','Contact
site',"$fn,$ln,$email,$mess");
      $output = '
      <p>Formulaire envoyé ! Merci pour votre
message !</p>';
    }
    return $output;
  }
}
```

Bob reteste ses modifications :

```
php index.php path='/contact'
php index.php path='/contact/post' fn='toto'
ln='toto' email='toto@toto.fr' mess=''
php index.php path='/contact/post' fn='toto'
ln='toto' email='toto@toto.fr' mess='test'
```

Tout fonctionne : il veut créer une nouvelle révision.

Vérifiez la branche courante (**feature/status**) :

```
git branch
```

Vérifiez votre index :

```
git status
```

Ajoutez les modifications :

```
git add .
```

Revérifiez votre index :

```
git status
```

Créez la nouvelle révision.

```
git commit -m "Ajout d'un statut global"
```

Affichez le journal sous forme de branche simplifiée :

```
git log --graph --oneline
```

Allez sur la branche dev

```
git switch dev
```

Affichez le journal sous forme de branche simplifiée :

```
git log --graph --oneline
```

Vous constatez que les journaux ne sont évidemment pas les mêmes, ce qui est normal : la branche **feature/status** n'est pas fusionnée pour le moment avec la branche **dev**. Bob choisit de fusionner sa branche.

Fusionner la branche **feature/status** dans la branche **dev** :

```
git merge feature/status
```

Affichez le journal sous forme de branche simplifiée :

```
git log --graph --oneline
```

Envoyez la branche dev dans le dépôt origin distant

```
git push origin dev
```

E.8) Gestion des formulaires

René entre dans la danse. Il doit réécrire le formulaire de contact et créer une classe globale qui prendra en charge la gestion des formulaires du site.

Allez dans *$PRJ/rene* :

```
cd $PRJ/rene
```

Ramenez l'ensemble du projet :

```
git clone $PRJ/le.depot.distant
```

Renommez *le.depot.distant* en *www.storcka.fr* :

```
# GNU/Linux
mv le.depot.distant www.storcka.fr
# Windows
move le.depot.distant www.storcka.fr
```

Allez dans *$PRJ/rene/www.storcka.fr* :

```
cd $PRJ/rene/www.storcka.fr
```

Configurez les identifiants GIT de René :

```
git config user.name René
git config user.email rene@storcka.fr
git config color.ui auto
```

Affichez les fichiers :

```
dir / ls
```

Au nombre de fichiers présents, on devine être sur la branche **master** du projet. En fait, toutes les branches distantes ont bien été clonées dans l'opération précédente.

Affichez les autres branches distantes disponibles :

```
git branch -r
```

Déplacez- vous sur branche **dev** :

```
git switch dev
```

Réaffichez les fichiers :

```
dir
ls
```

Affichez le journal :

```
git log
```

Vous remarquez que le clonage a intégré l'ensemble des révisions, ce qui n'est pas toujours utile dans la pratique.

Ainsi si vous rejoignez un projet important, il y a peu de chance que le code 10 ans en arrière vous intéresse...

Dans la pratique, il est alors plus intéressant (et raisonnable) de ne ramener qu'un nombre réduit de révisions avec l'option `--depth $NBR_DE_REVISIONS`, à rajouter à la commande `git clone` précédente...

Créez le fichier *core/form.class.php* :

```
<?php
class Form {
 public function __construct ($prop) {
   $this->prop = [
     'id' => '',
     'name' => '',
     'class' => '',
     'action' => '',
     'method' => 'post',
     'enctype' => 'multipart/form-data',
   ];
   $this->field = [];
   foreach ($prop as $key => $val) {
     $this->prop[$key] = $val;
   }
 }
 public function add ($field) {
   foreach ($field as $key => $val) {
     $this->field[$key] = $val;
     if (!isset($this->field[$key]['label'])) {
       $this->field[$key]['label'] = $key;
     }
     if (!isset($this->field[$key]['type'])) {
       $this->field[$key]['type'] = 'text';
     }
     if (!isset($this->field[$key]['val'])) {
       $this->field[$key]['val'] = '';
     }
     if (!isset($this->field[$key]['def'])) {
       $this->field[$key]['def'] = '';
     }
   }
 }
 public function check () {
   global $args, $_Status;
   reset($this->field);
   $ok1 = true;
   foreach ($this->field as $key => $val) {
     $ok2 = true;
     $data = trim($args[$key]);
     if (!mb_strlen($data)) {
       $ok2 &= $_Status->err('Le champ "'.
$val['label'].'" est vide !');
     }
     if (!strcmp('email',$key) && mb_strlen($data))
{
       if (!filter_var($data,
FILTER_VALIDATE_EMAIL)) {
         $ok2 &= $_Status->err('Le champ "'.
$val['label'].'" est erroné !');
       }
     }
     if ($ok2) { // no errors
       $this->set($key,$data);
     }
     $ok1 &= $ok2;
   }
   return $ok1;
 }
 public function set ($key, $data) {
   if (isset($this->field[$key])) {
     $this->field[$key]['val'] = $data;
   }
 }
```

```
 public function read () {
   $data = [];
   reset($this->field);
   foreach ($this->field as $key => $val) {
     $data[$key] = $val['val'];
   }
   return $data;
 }
 public function form () {
   $output = '';
   reset($this->field);
   foreach ($this->field as $key => $val) {
     $v = htmlentities(mb_strlen($val['val']) ?
$val['val'] : $val['def']);
     switch ($val['type']) {
       case 'text':
         $output .= '
     <div class="form_field">
       <p><label for="'.$key.'">'.
$val['label'].'</label> :</p>
       <p><input type="text" name="'.$key.'" id="'.
$key.'" value="'.$v.'" /></p>
     </div>';
         break;
       case 'textarea':
         $output .= '
     <div class="form_field">
       <p><label for="'.$key.'">'.
$val['label'].'</label> :</p>
       <p><textarea name="'.$key.'" id="'.
$key.'">'.$v.'</textarea></p>
     </div>';
         break;
     }
   }
   $form = '';
   reset($this->prop);
   foreach ($this->prop as $key => $val) {
     if (mb_strlen($val)) {
       $form .= " ".$key.'="'.$val.'"';
     }
   }
   return '
   <form'.$form.'>'.$output.'
     <div class="form_ctrl">
       <p><button
type="submit">Envoyer</button></p>
     </div>
   </form>';
 }
}
```

Modifiez le fichier *mod/fr.storcka.contact.class.php* :

```
<?php
namespace fr\storcka;
class Contact {
 public function __construct () {
   $this->_Form = new \Form([
     'id' => 'form_contact',
     'name' => 'form_contact',
     'action' => '/contact/post',
   ]);
   $this->_Form->add([
     'fn' => [
```

```
        'label' => 'Prénom',
      ],
      'ln' => [
        'label' => 'Nom',
      ],
      'email' => [
        'label' => 'Courriel',
      ],
      'mess' => [
        'label' => 'Message',
        'type' => 'textarea',
      ],
    ]);
  }
  public function form () {
    return $this->_Form->form();
  }
  public function handle () {
    $output = '';
    if (!$this->_Form->check()) { // some errors
occured...
      $output = $this->form();
    } else {
      $data = $this->_Form->read();
      // record datas somewhere...
      // $mess = print_r($data,true);
      // mail('contact@storcka.fr','Contact site',
$mess);
      $output = '
      <p>Formulaire envoyé ! Merci pour votre
message !</p>';
    }
    return $output;
  }
}
```

Vérifiez l'index et vérifiez notamment que vous êtes bien sur la branche **dev** :

```
git status
```

Ajoutez les fichiers :

```
git add .
```

Revérifiez l'index :

```
git status
```

Publiez les modifications en local :

```
git commit -m "Ajout d'une classe Form pour les
formulaires"
```

Envoyez les modifications sur la branche **dev** distante :

```
git push origin dev
```

Affichez le journal :

```
git log --oneline --graph
```

Remarquez cette fois que René n'a pas travaillé sur une branche **feature/** à part, mais directement sur la branche **dev**, ce qui n'est pas conseillé dans la pratique pour la raison suivante : imaginez plusieurs développeurs en train de travailler simultanément sur la branche **dev**. L'un d'eux veut publier ses modifs. Il récupère avec

`git pull` les modifications des autres, et commence à corriger les conflits avec son propre code. Mais pas de chance : pendant qu'il est en train de régler ses conflits, un autre développeur publie ses modifs sur **dev**, puis un autre... C'est vite le bordel pour tout le groupe de travail !

Donc en pratique, on travaille toujours sur des branches **feature/quelquechose** et c'est normalement au chef de projet, et à lui seul, de donner le feu vert pour la fusion dans la branche **dev** commune.

Cela étant, chaque société a sa manière de travailler, et les conseils ci-dessus ne valent pas pour Loi... Il s'agit juste de bien cerner les limites de GIT, et comprendre que le logiciel ne fera pas de miracle dans un environnement de type jungle ou bazar côté management...

E.9) Ajout d'un menu

Ce même jour, Alice a été chargée de rajouter une classe **Menu** et le menu par défaut du site. Pour le moment, elle n'a intégré aucun des changements de Bob et René, et se retrouve avec une version de code complètement dépassée.

Allez dans *$PRJ/alice/www.storcka.fr* :

```
cd $PRJ/alice/www.storcka.fr
```

Créez la branche **feature/menu** :

```
git branch
git branch feature/menu
git branch
```

Allez sur la branche :

```
git switch feature/menu
```

Modifiez *core/page.class.php* :

```
<?php
class Page {
 public static function header () {
   global $_Auth;
   header('Content-Type: text/plain; charset=UTF-
8');
   $_Menu = new Menu('menutop','path');
   $_Menu->add('/','Accueil');
   $_Menu->add('/contact','Contact');
   print '<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width,
initial-scale=1">
 <title>D\'Storcka</title>
</head>
<body>
 <header class="header">
   <div class="topbar">
     <div class="title">
       <img class="logo" src="img/logo/storcka.jpg"
alt="Logo D\'Storcka"/>
     </div>
     <nav class="nav">'.$_Menu->render('menutop').'
     </nav>
   </div>
```

```
  </header>
  <article>
    <div class="container">
';
    if (isset($_SESSION['auth/id']) &&
mb_strlen($_SESSION['auth/id'])) {
      print $_Auth->banner();
    }
  }
  public static function footer () {
    print '
    </div>
  <article>
</body>
</html>';
  }
}
```

Ajoutez *core/menu.class.php* :

```
<?php
class Menu {
  private $menu = null;
  public function __construct ($id,$var) {
    $this->menu = (object)[
      'id' => $id,
      'var' => $var,
      'uri' => [],
    ];
  }
  public function add ($path,$label) {
    $this->menu->uri[$path] = $label;
  }
  public function render ($id) {
    global $args;
    $li = '';
    $key = $this->menu->var;
    $val = isset($args[$key]) ? $args[$key] : '';
    reset($this->menu->uri);
    foreach ($this->menu->uri as $key2 => $val2) {
      $selected = !strcmp($val,$key2) ? '
class="selected"' : '';
      $li .= '
          <li><a href="'.$key2.'"'.$selected.'>'.
$val2.'</a></li>';
    }
    return '
        <ul id="'.$id.'">'.$li.'
        </ul>';
  }
}
```

Modifiez *core/uri.class.php* :

```
<?php
class URI {
  public function __construct () {
    $this->uri = [];
  }
  public function register ($path, $namespace,
$class, $func) {
    if (!isset($this->uri[$path])) {
      $this->uri[$path] = [
        'namespace' => $namespace,
        'class' => $class,
        'func' => $func,
```

```
      ];
    }
  }
  public function call ($path) {
    if (!isset($this->uri[$path])) {
      throw new Exception('Path '.$path .' not found
!');
    }
    $uri = $this->uri[$path];
    $n = $uri['namespace'];
    $c = $uri['class'];
    $f = $uri['func'];
    $nc = "$n\\$c";
    return (new $nc)->$f();
  }
  public function handle () {
    global $args;
    if (!isset($args['path'])) {
      $args['path'] = '/';
    }
    return $this->call($args['path']);
  }
}
```

Affichez le statut et vérifiez que vous êtes sur la branche **feature/menu** :

```
git status
```

Ajoutez les changements :

```
git add .
```

Réaffichez le status :

```
git status
```

Créez une révision :

```
git commit -m "Ajout d'une classe menu"
```

Se doutant qu'il y a eu des changements sur la branche **dev**, Alice veut mettre à jour le journal de la branche (sans ramener les fichiers pour le moment)

Mettez à jour l'index pour la branche dev :

```
git status
git fetch origin dev
git status
```

Allez sur la branche **dev** :

```
git switch dev
```

GIT prévient que la branche **dev** d'Alice est en retard de plusieurs révisions.

Affichez le journal actuel de la branche :

```
git log --oneline --graph
```

Alice s'aperçoit donc que les autres développeurs ont fait beaucoup de modifications.

Plutôt que de fusionner sa fonctionnalité à la branche **dev**, elle choisit la stratégie inverse : rebaser sa branche **feature/menu** à la dernière modification de la branche **dev**.

Cette stratégie lui permettra de résoudre les éventuels conflits de code dans sa branche **feature/menu** AVANT de la fusionner dans la branche **dev**.

Alice commence ainsi par mettre sa branche **dev** à jour :

```
git pull origin dev
```

Affichez le journal actuel de la branche :

```
git log --oneline --graph
```

Retournez sur sa branche **feature/menu**.

```
git switch feature/menu
```

Déplacez la base de la branche **feature/menu** sur la branche sur la branche **dev** fraîchement mise à jour :

```
git log --oneline –graph
git rebase dev
git log --oneline --graph
```

Ici il n'y aura pas de conflits dans l'opération, mais cela peut bien entendu arriver dans la pratique, surtout quand on est plusieurs développeurs travaillant sur une même branche...

Dans ce cas, GIT va reprendre toutes les révisions de la branche **feature/XXX**, une par une, en affichant les fichiers en conflit.

Le développeur devra alors, pour chaque révision, régler le conflit manuellement, faire un `git add`, puis un `git rebase --continue` pour passer à la révision suivante.

Dans les cas extrêmes, il peut aussi décider de tout abandonner en cours de route. Dans ce cas, il fera un `git rebase --abort`.

Comprenez bien, à cette étape, que les fichiers de la branche **dev** n'ont pas été affectés !

Nous avons toujours nos deux branches bien distinctes, simplement l'historique de la branche **feature/menu** a été réécrit pour refléter les changements opérés par les autres développeurs, et préparer à la fusion avec la branche **dev**.

L'avantage pour Alice, c'est qu'elle peut maintenant tester son code avec par exemple le formulaire de contact de Bob, chose qu'elle ne pouvait pas faire précédemment, parce que le formulaire de contact n'existait pas au moment où elle a commencé à coder sa fonctionnalité.

Vérifiez que vous êtes toujours sur la branche **feature/menu** :

```
git branch
```

Testez le formulaire de contact :

```
php index.php path='/contact'
```

Alice pourrait ainsi continuer à coder sur sa branche **feature/menu**, et à créer des révisions, tout en rebasant de temps à autres son code avec la branche **dev** pour ne pas cumuler trop de différences avec les autres développeurs.

Elle considère ici la fonctionnalité comme achevée, et choisit finalement de fusionner réellement **feature/menu** à la branche **dev**.

Comme ses fichiers sont désormais en avance sur la branche **dev**, la fusion se fera en mode avance rapide (**fast forward**).

Retournez sur la branche **dev** :

```
git switch dev
```

Affichez le journal actuel de la branche :

```
git log --oneline --graph
```

Fusionnez la branche **feature/menu** avec la branche **dev** :

```
git merge feature/menu
```

N.B. : remarquez bien l'absence de demande de commentaire, du à la fusion en mode avance rapide...

La fusion à coup de `rebase` inclus cependant une règle d'Or absolue : **on ne fait JAMAIS un `rebase` sur une branche publique partagée avec d'autres développeurs**.

En effet : le `rebase` reprend l'historique de la branche avec votre historique local, ce qui, au mieux, vous fera lyncher par les autres développeurs, au pire vous fera atomiser - au choix !

En outre, GIT ne vous laissera pas envoyer vos modifications à distance sans râler au moment du `push` vers le dépôt distant. Il vous demandera ainsi une confirmation particulière via l'option `--force`. Et là, il faut vraiment faire très attention àt ne pas se tromper ! Pas de `rebase` sur une branche partagée !

Dans la pratique, certains développeurs préfèrent faire un **rebase** avant le **merge** final parce que :

➔ ça évite une version intermédiaire de fusion

➔ ça permet d'entretenir un journal plus « linéaire », ne faisant pas apparaître les branches de développement temporaires

➔ ça permet de régler les conflits de code sur sa branche de développement

Inversement d'autres développeurs préfèrent ne pas faire de `rebase` et travaillent uniquement avec du `merge`, en imposant des informations de fusion, et en gardant les informations des branches de développement.

Bref, c'est un choix laissé à chacun d'adopter une vision ou l'autre, voir un mixe entre les deux suivant le contexte.

Mais en milieu pro, une bonne habitude quand on arrive sur un projet, c'est de faire un petit sondage auprès des autres développeurs sur les habitudes de la boîte...

Ici Alice choisit d'effacer sa branche **feature/menu**.

Effacez la branche **feature/menu** :

```
git branch -d feature/menu
```

Affichez le journal actuel de la branche :

```
git log --oneline --graph
```

Enfin, Alice publie ses changements sur la branche **dev** du dépôt distant :

```
git push origin dev
```

E.10) Première version officielle (release)

Après la réunion d'équipe, il a été décidé de fusionner la branche **dev** avec la branche **master** du projet.

Comme elle est la dernière à avoir fait des modifications dans le projet, Alice s'en charge et choisit une fusion en avance rapide.

Allez sur la branche master :

```
git switch master
```

Faites une fusion normale :

```
git merge dev
```

Affichez le journal actuel de la branche :

```
git log --oneline --graph
```

Envoyez la branche master sur le dépôt distant :

```
git push origin master
```

GIT indique qu'il nous manque en local des travaux de la branche distante (en fait, le fichier *index.html* de départ).

Mettez à jour la liste des fichiers/dossiers distants dans le cache local, et fusionnez les changements :

```
git ls-tree -r dev
git ls-tree -r remotes/origin/dev
git pull origin master
# Commentaire à modifier
Fusion de 'master' provenant du dépôt distant
git ls-tree -r dev
git ls-tree -r remotes/origin/dev
```

Si GIT indique qu'il ne trouve pas la méthode de fusion à utiliser, choisissez une fusion sans `rebase` ni `fast-foward` :

```
git config pull.rebase false
git pull origin master
# Commentaire à modifier
Fusion de 'master' provenant du dépôt distant
ls -l
```

Alice affiche la liste des fichiers de sa branche **master** :

```
tree
.
├── core
│   ├── auth.class.php
│   ├── const.inc.php
│   ├── form.class.php
│   ├── menu.class.php
│   ├── page.class.php
│   ├── status.class.php
│   └── uri.class.php
├── index.html
├── index.php
├── mod
│   ├── fr.storcka
│   │   ├── contact.class.php
│   │   ├── module.inc.php
│   │   ├── page
│   │   │   └── homepage.html
│   │   └── page.class.php
│   └── fr.storcka.contact.class.php
├── README.md
└── sandbox
    └── phpinfo.php
```

Elle peut enfin envoyer la branche master :

```
git push origin master
```

F) Alerte Générale !

C'est la panique ! Sans prévenir, l'hébergeur du site web a changé sa version de PHP, provoquant de nombreuses erreurs, dues aux évolutions du langage (classes dépréciées, nouvelles syntaxes, etc) !

Pas le choix : l'équipe installe en urgence la version PHP 8.2 et Bob est chargé de régler au plus vite les conflits de code dans une branche **hotfix**, avec de les propager sur **master** et **dev**.

Installez la version de PHP 8.2.

Si vous êtes sous Windows, modifiez vos variables d'environnement pour favoriser la version 8.2 avant la version 7.4 de PHP.

Si vous êtes sous GNU/Linux, sous **root** :

```
l
```

et choisissez la version 8.2.

Retournez sous Bob :

```
cd $PRJ/bob/www.storcka.fr
```

Allez sur la branche **dev** :

```
git switch dev
```

Récupérez les modifications :

```
git pull origin dev
```

Allez sur la branche master :

```
git switch master
```

Récupérez les modifications :

```
git pull origin master
```

Créez la branche **hotfix** depuis la branche **master** :

```
git branch
git branch hotfix
```

Et allez dessus :

```
git switch hotfix
```

N.B. : pour info, on aurait aussi pu faire les deux dernières commandes en une fois, avec `git checkout -b hotfix`.

Faites un

```
php index.php | less
```

Comme vous pouvez le voir au début de la sortie, plusieurs changements majeurs ont été réalisés dans les version 8.X de PHP, causant de nombreux problèmes avec les codes existants, qui doivent être revus et corrigés !

Bob va donc devoir corriger tous les scripts !

Modifiez *core/auth.class.php* :

```
<?php
```

```
class Auth {
 public function __construct () {
 }
 public static function form () : string {
   return '
   <div class="auth_popup">
     <form action="/" method="post"
enctype="multipart/form-data">
       <label for="login">Identifiant</label> :
<input type="text" name="login" id="login"
value=""/><br/>
       <label for="pass">Mot de passe</label> :
<input type="password" name="pass" id="pass"
value=""/><br/>
       <button type="submit">Envoyer</button>
     </form>
   </div>';
 }
 public function onstartup () : void {
   global $args;
   if (isset($args['login']) &&
isset($args['pass'])) {
     $_SESSION['auth/id'] = '';
     $_SESSION['auth/txt'] = '';
     if (!strcmp($args['login'],'james') && !
strcmp($args['pass'],'bond')) {
       $_SESSION['auth/id'] = '007';
       $_SESSION['auth/txt'] = 'James Bond';
     }
   }
 }
 public static function banner () : string {
   return '
   <div
class="auth_bar">'.htmlentities($_SESSION['auth/txt'
]).' <button type="button" id="disconnect">Se
déconnecter</button></div>
   ';
 }
 public static function onform () : string {
   global $args;
   if (!isset($_SESSION['auth/id']) || !
mb_strlen($_SESSION['auth/id'])) {
     if (isset($args['login']) ||
isset($args['pass'])) {
       print '<p class="error">Identifiants erronés
!</p>';
     }
     print Auth::form();
   } else {
     if (!ISCLI) {
       print '<p class="info">Bienvenue !</p>';
     }
   }
 }
}
```

Modifiez *core/const.inc.php* :

```
<?php
// for strtoupper and other string functions
setlocale(LC_ALL,'fr_FR.UTF8');
setlocale(LC_TIME,'fr_FR.UTF8');
setlocale(LC_CTYPE,'fr_FR.UTF8');
date_default_timezone_set('Europe/Paris');
```

```
// session
session_name('STORCKA');
session_start();
// for CLI debugging
define('ISCLI', PHP_SAPI === 'cli');
if (ISCLI) {
 parse_str(implode('&', array_slice($argv, 1)),
$_GET);
 $args = $_GET;
} else {
 $args = $_POST;
}
// errors
error_reporting(E_ALL ^ E_NOTICE);
// autoload classes
function autoload1 ($classname) {
 $target =
__DIR__.'/'.strtolower($classname).'.class.php';
 if (file_exists($target)) {
   include_once($target);
 }
}
spl_autoload_register('autoload1');
// Add status
$_Status = new Status();
// Add route mechanismus
global $_URI;
$_URI = new URI();
// Add modules
foreach (glob('mod/*/module.inc.php') as $key =>
$val) {
 include_once($val);
}
// authentication on startup
global $_Auth;
$_Auth  = new Auth();
$_Auth->onstartup();
```

Modifiez *core/form.class.php* :

```
<?php
class Form {
 private array $prop;
 private array $field;
 public function __construct (array $prop) {
   $this->prop = [
     'id' => '',
     'name' => '',
     'class' => '',
     'action' => '',
     'method' => 'post',
     'enctype' => 'multipart/form-data',
   ];
   foreach ($prop as $key => $val) {
     $this->prop[$key] = $val;
   }
   $this->field = [];
 }
 public function add (array $field) : void {
   foreach ($field as $key => $val) {
     $this->field[$key] = $val;
     if (!isset($this->field[$key]['label'])) {
       $this->field[$key]['label'] = $key;
     }
     if (!isset($this->field[$key]['type'])) {
```

```
        $this->field[$key]['type'] = 'text';
      }
      if (!isset($this->field[$key]['val'])) {
        $this->field[$key]['val'] = '';
      }
      if (!isset($this->field[$key]['def'])) {
        $this->field[$key]['def'] = '';
      }
    }
  }
  public function check () {
    global $args, $_Status;
    reset($this->field);
    $ok1 = true;
    foreach ($this->field as $key => $val) {
      $ok2 = true;
      if (!isset($args[$key]) || !
is_string($args[$key])) {
        $ok2 = $_Status->err('Le champ "'.
$val['label'].'" est invalide !');
      } else {
        $data = trim($args[$key]);
        if (!mb_strlen($data)) {
          $ok2 = $_Status->err('Le champ "'.
$val['label'].'" est vide !');
        } else {
          if (!strcmp('email',$key) &&
mb_strlen($data)) {
            if (!filter_var($data,
FILTER_VALIDATE_EMAIL)) {
              $ok2 = $_Status->err('Le champ "'.
$val['label'].'" est erroné !');
            }
          }
        }
      }
      if ($ok2) { // no errors
        $this->set($key,$data);
      }
      $ok1 &= $ok2;
    }
    return $ok1;
  }
  public function set (string $key, mixed $data) :
void {
    if (isset($this->field[$key])) {
      $this->field[$key]['val'] = $data;
    }
  }
  public function read () : array {
    $data = [];
    reset($this->field);
    foreach ($this->field as $key => $val) {
      $data[$key] = $val['val'];
    }
    return $data;
  }
  public function form () : string {
    $output = '';
    reset($this->field);
    foreach ($this->field as $key => $val) {
      $v = htmlentities(mb_strlen($val['val']) ?
$val['val'] : $val['def']);
      switch ($val['type']) {
```

```
        case 'text':
          $output .= '
      <div class="form_field">
        <p><label for="'.$key.'">'.
$val['label'].'</label> :</p>
        <p><input type="text" name="'.$key.'" id="'.
$key.'" value="'.$v.'" /></p>
      </div>';
          break;
        case 'textarea':
          $output .= '
      <div class="form_field">
        <p><label for="'.$key.'">'.
$val['label'].'</label> :</p>
        <p><textarea name="'.$key.'" id="'.
$key.'">'.$v.'</textarea></p>
      </div>';
          break;
      }
    }
    $form = '';
    reset($this->prop);
    foreach ($this->prop as $key => $val) {
      if (mb_strlen($val)) {
        $form .= " ".$key.'="'.$val.'"';
      }
    }
    return '
    <form'.$form.'>'.$output.'
      <div class="form_ctrl">
        <p><button
type="submit">Envoyer</button></p>
      </div>
    </form>';
  }
}
```

Modifiez *core/menu.class.php* :

```
<?php
class Menu {
 private object $menu;
 public function __construct (string $id, string
$var) {
    $this->menu = (object)[
      'id' => $id,
      'var' => $var,
      'uri' => [],
    ];
 }
 public function add (string $path, string $label) :
void {
    $this->menu->uri[$path] = $label;
 }
 public function render (string $id) : string {
    global $args;
    $li = '';
    $key = $this->menu->var;
    $val = isset($args[$key]) ? $args[$key] : '';
    reset($this->menu->uri);
    foreach ($this->menu->uri as $key2 => $val2) {
      $selected = !strcmp($val,$key2) ? '
class="selected"' : '';
      $li .= '
```

```
          <li><a href="'.$key2.'"'.$selected.'>'.
$val2.'</a></li>';
    }
    return '
        <ul id="'.$id.'">'.$li.'
        </ul>';
  }
}
```

Modifiez *core/page.class.php* :

```
<?php
class Page {
 public static function header () : void {
   global $_Auth, $_Status;
   header('Content-Type: text/plain; charset=UTF-
8');
   $_Menu = new Menu('menutop','path');
   $_Menu->add('/','Accueil');
   $_Menu->add('/contact','Contact');
   print '<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width,
initial-scale=1">
 <title>D\'Storcka</title>
</head>
<body>
 <header class="header">
   <div class="topbar">
     <div class="title">
       <img class="logo" src="img/logo/storcka.jpg"
alt="D\'Storcka"/>
     </div>
     <nav class="nav">'.$_Menu->render('menutop').'
     </nav>
   </div>
 </header>
 <article>
   <div class="container">
';
   if (isset($_SESSION['auth/id']) &&
mb_strlen($_SESSION['auth/id'])) {
     print $_Auth->banner();
   }
   print $_Status->html();
 }
 public static function footer () : void {
   print '
   </div>
 <article>
</body>
</html>';
 }
}
```

Modifiez *core/status.class.php* :

```
<?php
class Status {
 private array $mess;
 public function __construct () {
   $this->mess = [];
 }
```

```
 private function add (string $type, string $mess) :
void {
   if (!isset($this->mess[$type])) {
     $this->mess[$type] = [];
   }
   $this->mess[$type][] = $mess;
 }
 public function err (string $mess) : bool {
   $this->add('error',$mess);
   return false;
 }
 public function info (string $mess) : bool {
   $this->add('info',$mess);
   return true;
 }
 public function warn (string $mess) : bool {
   $this->add('warn',$mess);
   return true;
 }
 public function html () : string {
   reset($this->mess);
   $output = '';
   foreach ($this->mess as $key => $val) {
     if (count($val)) {
       foreach ($val as $key2 => $val2) {
         $output .= '
     <p class="'.$key.'">'.$val2.'</p>';
       }
     }
   }
   return $output;
 }
```

```
}
```

Modifiez *core/uri.class.php* :

```
<?php
class URI {
 private array $uri;
 public function __construct () {
   $this->uri = [];
 }
 public function register (string $path, string
$namespace, string $class, string $func) : void {
   if (!isset($this->uri[$path])) {
     $this->uri[$path] = [
       'namespace' => $namespace,
       'class' => $class,
       'func' => $func,
     ];
   }
 }
 public function call (string $path) : mixed {
   if (!isset($this->uri[$path])) {
     throw new Exception('Path '.$path .' not found
!');
   }
   $uri = $this->uri[$path];
   $n = $uri['namespace'];
   $c = $uri['class'];
   $f = $uri['func'];
   $nc = "$n\\$c";
   return (new $nc)->$f();
 }
 public function handle () : mixed {
```

```
    global $args;
    if (!isset($args['path'])) {
      $args['path'] = '/';
    }
    return $this->call($args['path']);
  }
}
```

Modifiez *mod/fr.storcka/contact.class.php* :

```
<?php
namespace fr\storcka;
class Contact {
 private \Form $_Form;
 public function __construct () {
    $this->_Form = new \Form([
      'id' => 'form_contact',
      'name' => 'form_contact',
      'action' => '/contact/post',
    ]);
    $this->_Form->add([
      'fn' => [
        'label' => 'Prénom',
      ],
      'ln' => [
        'label' => 'Nom',
      ],
      'email' => [
        'label' => 'Courriel',
      ],
      'mess' => [
        'label' => 'Message',
        'type' => 'textarea',
      ],
    ]);
  }
 public function form () : string {
    return $this->_Form->form();
  }
 public function handle () : string {
    $output = '';
    if (!$this->_Form->check()) { // some errors
occured...
      $output = $this->form();
    } else {
      $data = $this->_Form->read();
      // record datas somewhere...
      // $mess = print_r($data,true);
      // mail('contact@storcka.fr','Contact site',
$mess);
      $output = '
      <p>Formulaire envoyé ! Merci pour votre
message !</p>';
    }
    return $output;
  }
}
```

Modifiez *mod/fr.storcka/page.class.php* :

```
<?php
namespace fr\storcka;
class Page {
 public function __construct () {
 }
 public function php (string $pagename) : string {
```

```
    $file = __DIR__.'/page/'.$pagename.'.php';
    if (file_exists($file)) {
      ob_start();
      include_once($file);
      $res = ob_get_contents();
      ob_end_clean();
      return $res;
    }
    return '';
  }
  public function html (string $pagename) : string {
    $file = __DIR__.'/page/'.$pagename.'.html';
    if (file_exists($file)) {
      return file_get_contents($file);
    }
  }
  public function call (string $pagename) : string {
    //print "calling $pagename...\n";
    $output = '';
    $output .= $this->php($pagename);
    $output .= $this->html($pagename);
    return $output;
  }
  public function handle () : string {
    global $args;
    if (!isset($args['page'])) {
      return $this->call('homepage');
    }
    return $this->call($args['page']);
  }
}
```

Après de si gros changements dans le code, Bob veut revérifier les principales fonctionnalités :

```
php index.php
php index.php path="/contact"
php index.php path="/contact/post" fn="toto"
ln="toto" mess="toto"
php index.php path="/contact/post" fn="toto"
ln="toto" email="toto" mess="toto"
php index.php path="/contact/post" fn="toto"
ln="toto" email="toto@toto.fr" mess="toto"
php index.php login="james" pass="bond"
php index.php login="james" pass="bond"
path="/contact"
```

Apparemment tout est ok.

La structure est :

```
tree
.
├── core
│   ├── auth.class.php
│   ├── const.inc.php
│   ├── form.class.php
│   ├── menu.class.php
│   ├── page.class.php
│   ├── status.class.php
│   └── uri.class.php
├── index.html
├── index.php
├── mod
│   ├── fr.storcka
│   │   ├── contact.class.php
```

```
│   │   ├── module.inc.php
│   │   ├── page
│   │   │   └── homepage.html
│   │   └── page.class.php
│   └── fr.storcka.contact.class.php
└── README.md
```

Mais pendant qu'il corrigeait les scripts, René, qui n'a pas encore fait la mise à jour en PHP 8.X, a été chargé d'améliorer la classe URI du Core : Bob et Alice ont en effet oublié de rajouter dans les paramètres la page parente et un label pour chaque URI, qui servira plus tard à la construction du fil d'Ariane (ou breadcrumbs).

ATTENTION : à partir de maintenant, c'est à vous de trouver et de compléter les commandes indiquées par trois points de suspension !

Allez sous *$PRJ/rene/www.storcka.fr* :

```
cd ...
```

Allez sur la branche **dev** :

```
git s.. d..
```

Mettez la branche à jour depuis le dépôt **origin** :

```
git p.. o.. d..
```

Modifiez *core/uri.class.php* :

```
<?php
class URI {
 public function __construct () {
   $this->uri = [];
 }
 public function register ($parentpath, $path,
$title, $namespace, $class, $func) {
   if (!isset($this->uri[$path])) {
     $this->uri[$path] = [
       'parentpath' => $parentpath,
       'title' => $title,
       'namespace' => $namespace,
       'class' => $class,
       'func' => $func,
     ];
   }
 }
 public function call ($path) {
   if (!isset($this->uri[$path])) {
     throw new Exception('Path '.$path .' not found
!');
   }
   $uri = $this->uri[$path];
   $n = $uri['namespace'];
   $c = $uri['class'];
   $f = $uri['func'];
   $nc = "$n\\$c";
   return (new $nc)->$f();
 }
 public function handle () {
   global $args;
   if (!isset($args['path'])) {
     $args['path'] = '/';
   }
   return $this->call($args['path']);
 }
```

```
}
```

Modifiez *mod/fr.storcka/module.inc.php* :

```
<?php
global $_URI;
include_once(__DIR__.'/page.class.php');
$_URI->register(null, '/','Accueil','fr\
storcka','Page','handle');
$_URI->register(null, '/index.html','Accueil','fr\
storcka','Page','handle');
$_URI->register(null, '/index.php','Accueil','fr\
storcka','Page','handle');
include_once(__DIR__.'/contact.class.php');
$_URI->register('/','/contact','Contact','fr\
storcka','Contact','form');
$_URI->register('/','/contact/post','Contact','fr\
storcka','Contact','handle');
```

René met à jour son index local :

```
git s..
git a.. .
git s..
```

Et comme il est plus rapide que Bob, il poste ses changements avec l'ancien format sur la branche **dev** du dépôt distant.

```
git p.. o.. d..
```

Et là, c'est le drame : GIT renvoie un **"Everything up-to-date"** mais sans poster les changements ?! Un `git status` confirme que les fichiers sont toujours prêts à être envoyés depuis la zone de transfert vers le dépôt **origin** distant, mais ils ne partent pas !

René commence par revérifier son dépôt :

```
git remote show origin
```

Ce n'est apparemment pas le problème.

Il consulte le journal :

```
git s..
```

et se rend alors compte qu'il a commis deux énormes erreurs :

➔ d'abord il a OSÉ travailler sur la branche **dev** commune directement, au lieu de créer une **feature/XXX**…

➔ ensuite il a complètement oublié de créer une révision via :

```
git commit -m "Ajout des champs parentpath et title
dans core/uri.class.php"
```

qu'il se dépêche de faire ici.

Heureusement pour lui, ici, ces petites erreurs n'auront pas plus de conséquences ici, mais comme nous l'avons dit tantôt, ce n'est pas bien de travailler directement sur **dev**...

Exécutez donc la commande précédente.

Retentez l'envoi de la branche vers **origin** :

```
git p... o... d...
git s..
```

Cette fois la branche **dev** a bien été envoyée !

MAIS gardez bien à l'esprit que son code source n'est toujours pas compatible PHP 8.x...

Revenons à Bob, toujours attelé aux modifications du code vers PHP 8.x, et toujours sur sa branche **hotfix**, prêt à envoyer ses premiers changements, à la fois sur la branche **master**, mais aussi sur la branche **dev** !

```
cd $PRJ/b../w..
```

Bob met à jour son dépôt local :

```
git s..
git a.. .
git s..
git c.. -m "hotfix passage à PHP8.X"
```

Il retourne sur la branche **master**, et décide de forcer une révision pour laisser une trace avec l'option `--no-ff` (no fast forward)

```
git s.. m..
git m.. h.. --no-ff
```

Dans le commentaire demandé, il écrit :

```
Corrections suite au passage à PHP 8.X
```

Il envoie d'abord la branche master corrigée

```
git p.. o.. m..
```

Il ne lui reste plus qu'à envoyer le **hotfix** sur la branche **dev**.

Allez sur la branche **dev** locale :

```
git s.. d..
```

Bob fait une fusion de hotfix en mode avance rapide.

Fusionnez la branche **hotfix** dans **dev** :

```
git m.. h..
```

Vérifiez votre index :

```
git s..
```

GIT indique que, dans le cache local, la branche **dev** est en avance sur **origin/dev** de 3 révisions.

Envoyez la branche **dev** sur le dépôt **origin** :

```
git p.. o.. d..
```

Forcément, René ayant modifié la branche avant Bob, GIT refuse l'envoi… Comprenez ici que le cache local ne reflète pas fidèlement le dépôt distant.

Bob n'a donc pas le choix : il doit d'abord intégrer les changements de René, encore en PHP 7.x, en les corrigeant, bien entendu...

Ramenez la branche **dev** distante en local

```
git p.. o.. d..
```

GIT indique un conflit de fusion dans core.uri.class.php. Réglez-le rapidement de :

```
...
<<<<<<< HEAD
 public function register (string $path, string
$namespace, string $class, string $func) : void {
=======
```

```
 public function register ($parentpath, $path,
$title, $namespace, $class, $func) {
>>>>>>> 99de2fa934749733a0bc84f777f45618c21c5bb9
...
```

à :

```
 public function register (string $parentpath,
string $path, string $title, string $namespace,
string $class, string $func) : void {
```

Vérifiez votre index et faites une révision :

```
git s..
git c.. -m "Fusion après hotfix"
```

GIT vous affiche l'erreur :

```
error: Impossible de valider car vous avez des
fichiers non fusionnés.
```

Et oui… Bon a oublié d'utiliser git add pour valider le fichier fusionné ! Il corrige donc son erreur et retente sa révision :

```
git add .
git s
git c.. -m "Fusion après hotfix"
```

Cette fois la révision ne produit aucune erreur.

Bon envoie donc ses modifications de **dev** vers **origin** :

```
git p.. o.. d..
```

F.1) Revue de l'authentification

Alice reçoit une nouvelle mission : rendre la classe **Auth** non statique.

Elle n'a pas encore eu le temps de voir les changements entre PHP7 et PHP8.

Elle commence donc par ramener les changements opérés par ses collègues sur sa branche **dev**.

Déplacez-vous sur la branche **dev** et mettez-la à jour.

```
cd $PRJ/a../w..
git s.. d..
git log --oneline --graph
git p.. o.. d..
git log --oneline --graph
```

Alice veut voir toutes les différences entre l'avant-avant dernière révision et le code actuel en PHP8 :

```
git diff HEAD^^ HEAD
```

Comme il y a trop de classes modifiées et d'informations, elle décide de se focaliser sur la seule classe **Auth**.

```
git diff HEAD^^:core/uri.class.php
HEAD:core/uri.class.php
```

Une autre manière de faire est de rechercher d'abord toutes les révisions contenant le fichier :

```
git log --oneline -graph core/uri.class.php
```

Puis de demander la différence entre le code actuel et l'ancienne révision :

```
git diff $REVISION core/uri.class.php
```

Remarquez bien ici que le double ancêtre direct `HEAD^^` n'est pas la même chose que de prendre l'avant-avant dernière révision affichée dans le journal, laquelle pointe vers une autre branche !

Alice commence à recoder sa fonctionnalité **feature/auth2** :

```
git c.. -b feature/auth2
```

Modifiez le fichier *core/auth.class.php* :

```
<?php
class Auth {
 public function __construct () {
 }
 public function form () : string {
   return '
   <div class="auth_popup">
     <form action="/" method="post"
enctype="multipart/form-data">
       <label for="login">Identifiant</label> :
<input type="text" name="login" id="login"
value=""/><br/>
       <label for="pass">Mot de passe</label> :
<input type="password" name="pass" id="pass"
value=""/><br/>
       <button type="submit">Envoyer</button>
     </form>
   </div>';
 }
 private function disconnect () : void {
   $_SESSION['auth/id'] = '';
   $_SESSION['auth/txt'] = '';
   $_SESSION['auth/connected'] = false;
   $_SESSION['auth/usr/tpl'] = 'default';
 }
 private function connect (string $id, string
$txt) : void {
   $_SESSION['auth/id'] = $id;
   $_SESSION['auth/txt'] = $txt;
   $_SESSION['auth/connected'] = true;
   $_SESSION['auth/usr/tpl'] = 'default';
 }
 public function onstartup () : void {
   global $args;
   if (!isset($_SESSION['auth/connected'])) {
     $this->disconnect();
   }
   if (isset($args['login']) &&
isset($args['pass'])) {
     $this->disconnect();
     if (!strcmp($args['login'],'james') && !
strcmp($args['pass'],'bond')) {
       $this->connect('007','James Bond');
     }
   }
 }
 public function banner () : string {
   return '
   <div
class="auth_bar">'.htmlentities($_SESSION['auth/txt'
```

```
]).' <button type="button" id="disconnect">Se
déconnecter</button></div>
    ';
  }
  public function onform () : string {
    global $args;
    $output = '';
    if (!$_SESSION['auth/connected']) {
      if (isset($args['login']) ||
isset($args['pass'])) {
        $output .= '<p class="error">Identifiants
erronés !</p>';
      }
      $output .= $this->form();
    } else {
      if (!ISCLI) {
        $output .= '<p class="info">Bienvenue
!</p>';
      }
    }
    return $output;
  }
}
```

Alors qu'elle est en train de travailler encore sur ses tests, René, en déplacement, lui téléphone pour lui signaler une erreur dans le fichier *core/uri.class.php*, et lui demande de corriger en urgence.

Alice ne veut évidemment pas perdre ses changements en cours. **Elle va donc remiser temporairement son code**, et s'occuper du problème signalé.

Remisez le code :

```
git status
git stash save "travaux sur auth.class.php"
git status
```

Remarquez que pour le moment, elle reste sur **feature/auth2**.

Modifiez *core/uri.class.php* :

```
<?php
class URI {
  private array $uri;
  public function __construct () {
    $this->uri = [];
  }
  public function register (mixed $parentpath, string
$path, string $title, string $namespace, string
$class, string $func) : void {
    if (!isset($this->uri[$path])) {
      $this->uri[$path] = [
        'parentpath' => $parentpath,
        'title' => $title,
        'namespace' => $namespace,
        'class' => $class,
        'func' => $func,
      ];
    }
  }
  public function call (string $path) : mixed {
    if (!isset($this->uri[$path])) {
      throw new Exception('Path '.$path .' not found
!');
    }
    $uri = $this->uri[$path];
```

```
    $n = $uri['namespace'];
    $c = $uri['class'];
    $f = $uri['func'];
    $nc = "$n\\$c";
    return (new $nc)->$f();
  }
  public function handle () : mixed {
    global $args;
    if (!isset($args['path'])) {
      $args['path'] = '/';
    }
    return $this->call($args['path']);
  }
}
```

Alice se pose la question de savoir s'il y a eu des changements sur la branche dev du dépôt origin, autrement dit si un de ses collègues à poster des modifications depuis qu'elle a ramené la branche en local :

```
git diff remotes/origin/dev dev
```

Comme personne n'a posté de changements, et qu'Alice est pressée, elle décide de faire sa révision directement sur la branche **dev**.

Déplacez-vous sur la branche :

```
git b...
git s... d...
git b...
```

Ajoutez les fichiers modifiés à l'index.

```
git a.. .
git s...
```

Et remarquez bien que le code remisé n'apparaît pas dans l'index ! Terminez la révision et l'envoi du code sur le dépôt distant :

```
git c... -m "Hotfix de core/uri.class.php sur
demande téléphonique de René"
git push origin dev
```

Alice a fini sa modification, elle revient sur sa branche de travail et ramène ses modifications :

```
git s... feature/auth2
git stash list
git stash pop
```

En faisant un :

```
php index.php
```

Alice remarque que le bug affecte logiquement aussi sa version. Elle doit donc ramener le correctif fraîchement réalisé sur la branche **dev**. Elle décide logiquement de faire un `rebase`.

```
git rebase dev
```

GIT lui signale qu'elle a fait des modifications encore non validées, et refuse logiquement le `rebase`.

Ajoutez donc le fichier modifié à l'index, et faites une révision :

```
git s..
git a.. .
git s..
```

```
git c.. -m "Réécriture des méthodes de
auth.class.php en dynamique"
git rebase dev
php index.php login="james" pass="bond"
```

Tout fonctionne à nouveau.

À peine terminée sa modification, Alice reçoit la mission de modifier le *core/status.class.php* pour y intégrer une sortie texte.

Elle n'a pas encore verser ses changements dans la branche **dev**, et plutôt que de créer une nouvelle branche, elle choisit de continuer sur celle ou elle se trouve.

Modifiez la classe *core/status.class.php* :

```
<?php
class Status {
 private array $mess;
 public function __construct () {
   $this->mess = [];
 }
 private function add (string $type, string $mess) :
void {
   if (!isset($this->mess[$type])) {
     $this->mess[$type] = [];
   }
   $this->mess[$type][] = $mess;
 }
 public function err (string $mess) : bool {
   $this->add('error',$mess);
   return false;
 }
 public function info (string $mess) : bool {
   $this->add('info',$mess);
   return true;
 }
 public function warn (string $mess) : bool {
   $this->add('warn',$mess);
   return true;
 }
 public function html () : string {
   reset($this->mess);
   $output = '';
   foreach ($this->mess as $key => $val) {
     if (count($val)) {
       foreach ($val as $key2 => $val2) {
         $output .= '
     <p class="'.$key.'">'.$val2.'</p>';
       }
     }
   }
   return $output;
 }
 public function text () : string {
   reset($this->mess);
   $output = '';
   foreach ($this->mess as $key => $val) {
     if (count($val)) {
       foreach ($val as $key2 => $val2) {
         $output .= "\n".$val2;
       }
     }
   }
   return $output;
```

```
  }
}
```

Mais rebelote : à peine rentré, René signale une autre coquille urgente, cette fois dans *core/menu.class.php* ! Alice remise donc une nouvelle fois son code.

```
git status
git stash save "Modif de core/status.class.php non
validée"
git status
```

Modifiez le fichier *core/menu.class.php* :

```
<?php
class Menu {
 private object $menu;
 public function __construct (string $id, string
$class, string $var) {
    $this->menu = (object)[
      'id' => $id,
      'class' => $class,
      'var' => $var,
      'uri' => [],
    ];
 }
 public function add (string $path, string $label) :
void {
    $this->menu->uri[$path] = $label;
 }
 public function render (string $id) : string {
    global $args;
    $li = '';
    $key = $this->menu->var;
    $val = isset($args[$key]) ? $args[$key] : '';
    reset($this->menu->uri);
    foreach ($this->menu->uri as $key2 => $val2) {
      $selected = !strcmp($val,$key2) ? '
class="selected"' : '';
      $li .= '
          <li><a href="'.$key2.'"'.$selected.'>'.
$val2.'</a></li>';
    }
    return '
        <ul id="'.$this->menu->id.'" class="'.$this-
>menu->class.'">'.$li.'
        </ul>';
 }
}
```

Le menu étant appelé dans la classe page, modifiez *core/page.class.php* :

```
<?php
class Page {
 public static function header () : void {
    global $_Auth, $_Status;
    header('Content-Type: text/plain; charset=UTF-
8');
    $_Menu = new Menu('menutop','menutop','path');
    $_Menu->add('/','Accueil');
    $_Menu->add('/contact','Contact');
    print '<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
```

```
 <meta name="viewport" content="width=device-width,
initial-scale=1">
 <title>D\'Storcka</title>
</head>
<body>
 <header class="header">
   <div class="topbar">
     <div class="title">
       <img class="logo" src="img/logo/storcka.jpg"
alt="D\'Storcka"/>
     </div>
     <nav class="nav">'.$_Menu->render('menutop').'
     </nav>
   </div>
 </header>
 <article>
   <div class="container">
';
   if (isset($_SESSION['auth/id']) &&
mb_strlen($_SESSION['auth/id'])) {
     print $_Auth->banner();
   }
   print $_Status->html();
 }
 public static function footer () : void {
   print '
   </div>
 <article>
</body>
</html>';
 }
}
```

Pas de chance : c'est au tour de Bob cette fois de demander l'ajout prioritaire d'une classe spéciale pour gérer les scripts Javascript !

De nouveau Alice remise donc ses changements, en restant sur sa branche.

```
git status
git stash list
git stash save "changements dans core/page/class.php
et core/menu.class.php"
git stash list
git status
```

Créez *core/js.class.php* :

```
<?php
class JS {
 private array $src;
 public function __construct () {
   $this->src = [];
 }
 public function add (string $src, int $priority) :
void {
   $this->src[] = [
     'src' => $src,
     'priority' => $priority,
   ];
 }
 public function row ($src) : string {
       return '
   <script type="text/javascript" src="'.
$src.'"></script>';
 }
 public function render () : string {
```

```
    $output = '';
    usort($this->src, function ($item1, $item2) {
      return $item1['priority']-$item2['priority'];
    });
    foreach ($this->src as $key2 => $val2) {
      $output .= $this->row($val2['src']);
    }
    return $output;
  }
}
```

Alice allait poster ses changements, quand soudainement le patron arrive dans le bureau, et après discussion, considère que les modifications demandées par Bob et René ne sont pas prioritaires sur son travail (« Si vous avez toujours raison, soit vous êtes quelqu'un de vraiment formidable, soit vous êtes le patron... »).

Alice doit donc reprendre sa modification, restée en suspend, et l'intégrer en urgence à la branche **dev** en priorité.

Alice commence donc par remiser le code en cours.

```
git status
git stash save "ajout de js.class.php sur demande
Bob"
git status
```

Et là, c'est le drame (le retour) : GIT refuse de remiser le fichier en attente, ce qui est parfaitement normal, puisque le fichier n'est pas encore « suivi »...

Il faut dont faire un :

```
git status
git add .
git stash save "ajout de js.class.php sur demande
Bob"
git status
```

N.B.: on aurait aussi pu utiliser ici `git stash -u`, le `-u` demandant à GIT de prendre les fichiers non suivis (untracked). Une autre option est `git stash -a` (all), qui reprend l'option `-u` en y ajoutant les fichiers ignorés.

Alice veut ensuite revoir la liste de ses codes laissés en suspend, qui commence à grossir dangereusement...

```
git stash list
```

Remarquez que GIT indique la branche sur laquelle chaque remisage a été fait, ce qui permet donc de cumuler plusieurs remisages sur plusieurs branches, en toute sécurité. Remarquez également le sens de la pile de remisage : le dernier remisage commence à l'indice 0.

On peut revoir les changements d'un code suspendu.

Affichez les changements des différents codes remisés

```
git stash show
git stash show -p
git stash show stash@{1}
git stash show -p stash@{1}
git stash show stash@{2}
git stash show -p stash@{2}
```

Alice doit maintenant réintégrer les changements de son premier remisage (donc à l'indice 2) :

```
git status
git stash pop stash@{2}
git status
```

N.B.: l'option `pop` est l'équivalent des options `apply` (appliquer) + `drop` (effacer), qui utilisent les mêmes syntaxes.

Alice envoie sa modification de *core/status.class.php* sur la branche **dev** et dans le dépôt **origin**.

```
git s.. d..
git p.. o.. d..
git s..
git a.. .
git s..
git commit -m "Modification de
core/status.class.php"
git p.. o.. d..
```

Et elle retourne sur sa branche **feature/auth2** :

```
git s.. f..
```

Elle vérifie sa pile de code remisé :

```
git stash list
```

Maintenant qu'elle a fini l'urgence, Alice reprend les codes de René et Bob laissés en suspend.

```
git stash pop stash@{1}
git stash pop
```

P.S.: si les remisages sont tous indépendants les uns des autres, ce qui est ici le cas, il n'y a pas de conflits possibles : on peut donc faire du pop à priori sans risques. En revanche, si vous avez remisé plusieurs fois un même fichier, faites bien attention à l'ordre des pop : le dernier écrasera les précédents !

Autre possibilité non exploitée dans cet exemple : envoyer un code remisé sur une autre branche : `git stash branch $NOM stash@{1}` par exemple.

Enfin, si vous désirez enlever tous les codes remisés en attente : `git stash clear` est votre ami...

Alice a fini ses modifications. Elle fait donc une révision finale :

```
git s..
git a.. .
git s..
git commit -m "Modification de
core/menu.class.php+page.class.php et ajout de
core/js.class.php sur demandes Bob+René"
```

Elle fusionne ses changements avec la branche **dev**, qu'elle envoie sur le dépôt distant :

```
git s.. d..
git p.. o.. d..
git s..
git m.. f..
```

avec le commentaire :

```
Modification de core/menu.class.php+page.class.php
et ajout de core/js.class.php sur demandes Bob+René
git s..
git log
```

Alice n'a pas fait attention : le commentaire de fusion est le même que le commentaire contenu dans la révision de la branche (donc doublon). Elle modifie son dernier commentaire avec

```
git commit --amend
```

et le commentaire raccourci :

```
Modifications demandées par Bob & René
```

Vérifiez le changement :

```
git log
```

Envoyez les changements sur le dépôt **origin** :

```
git p.. o.. d..
```

Effacez la branche **feature/auth** devenue inutile :

```
git b.. -.. f..
```

F.2) Ajout de templates (modèles)

Pendant de temps-là, Bob a reçu la mission de revoir le système de modules, en intégrant un mécanisme de modèles HTML (templates).

Retournez sous Bob :

```
cd ...
```

Créez une branche **feature/modules** et allez dessus :

```
gir c... -b f...
```

Créez le dossier *tpl/default* puis le fichier *tpl/default/tpl.class.php* :

```
<?php
namespace tpl\default;
class Tpl extends \Tpl_HTML {
 public function __construct () {
   parent::__construct();
   $this->html = '<!DOCTYPE html>
<html lang="fr">
<head>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width,
initial-scale=1">
 <title>{{title}}</title>
 {{{head/js}}}
 {{{head/css}}}
</head>
<body>
 <header class="header">
   <div class="topbar">
     <div class="title">
       <img class="logo"
src="tpl/default/img/logo/storcka.jpg"
alt="D\'Storcka"/>
     </div>
     <nav class="nav">{{{top/bar/menu}}}
     </nav>
   </div>
   <div class="left">{{{header/left}}}</div>
   <div class="center">{{{header/center}}}</div>
   <div class="right">{{{header/right}}}</div>
 </header>
 <article>
```

```
    <div class="body container">
      <div class="left">{{{body/left}}}</div>
      <div class="center">{{{body/center}}}</div>
      <div class="right">{{{body/right}}}</div>
    </div>
  </article>
  <footer class="footer">
    <div class="left">{{{footer/left}}}</div>
    <div class="center">{{{footer/center}}}</div>
    <div class="right">{{{footer/right}}}</div>
  </footer>
  {{foot/js}}
</body>
</html>';
  }
  public function init () {
    $this->discover();
    $this->add_header('Content-Type: text/plain;
charset=UTF-8');
    $this->var['title'] = 'D\'Storcka';
    // will be populate by slot.inc.php
    $_Menu = new \Menu('menutop','menutop','path');
    $this->obj['menutop'] = $_Menu;
    // will be populate by slot.inc.php
    $_JS = new \JS();
    $_JS->add('js/jquery.js',10);
    $_JS->add('js/main.js',20);
    $this->obj['jstop'] = $_JS;
    //$_CSS = new CSS();
    //$_CSS->add('css/main.css',10);
    //$this->obj['css'] = $_CSS;
  }
```

```
  public function merge ($data) {
    global $_Auth, $_Status;
    $this->add('head/js',$this->obj['jstop']-
>render(),10);
    //$this->add('head/css',$this->obj['css']-
>render(),10);
    $this->add('top/bar/menu',$this->obj['menutop']-
>render('menutop'),10);
    if ($_SESSION['auth/connected']) {
      $this->add('body/center',$_Auth->banner(),10);
    }
    $this->add('body/center',$_Status->html(),20);
    $this->add('body/center',$data,30);
  }
}
```

Ajoutez le fichier à l'index, et faites une révision :

```
git s..
git add .
git s..
git c.. -m "Ajout d'un template"
```

Créez *core/mod.class.php* :

```
<?php
class Mod {
  public string $name;
  public string $path;
  public string $namespace;
  public bool $enabled;
  public function __construct () {
    $this->name = '';
```

```
    $this->path = '';
    $this->namespace = '';
    $this->enabled = false;
  }
  public function register (string $name, string
$path, string $namespace) {
    global $mod;
    $this->name = $name;
    $this->path = $path;
    $this->namespace = $namespace;
    $mod[$namespace] = $this;
  }
  public function slot (string $action) {
    $func = $this->namespace.'\on';
    if (!function_exists($func)) {
      throw new Exception('Missing function "'.
$func.'" !');
    }
    $func($action);
  }
  public function enable () {
    $this->slot('enabled');
    $this->enabled = true;
  }
  public function disable () {
    $this->slot('disabled');
    $this->enabled = false;
  }
}
```

Ajoutez le fichier à l'index, et faites une révision :

```
git s..
```

```
git add .
git s..
git c.. -m "Ajout de la classe mod.class.php"
```

C'est le soir. Bob est fatigué. Il termine sa journée en allant sur la branche **master**, pour y voir si des changements ont été apportés.

```
git s.. master
git fetch origin master
```

Le lendemain arrive et Bob continue son intégration.

Modifiez *core/const.inc.php* :

```
<?php
// for strtoupper and other string functions
setlocale(LC_ALL,'fr_FR.UTF8');
setlocale(LC_TIME,'fr_FR.UTF8');
setlocale(LC_CTYPE,'fr_FR.UTF8');
date_default_timezone_set('Europe/Paris');
// session
session_name('STORCKA');
session_start();
// for CLI debugging
define('ISCLI', PHP_SAPI === 'cli');
if (ISCLI) {
 parse_str(implode('&', array_slice($argv, 1)),
$_GET);
 $args = $_GET;
} else {
 $args = $_POST;
}
// errors
```

```
error_reporting(E_ALL ^ E_NOTICE);
// autoload classes
function autoload1 ($classname) {
 $target =
__DIR__.'/'.strtolower($classname).'.class.php';
 if (file_exists($target)) {
   include_once($target);
 }
}
spl_autoload_register('autoload1');
// Add status
$_Status = new Status();
// Modules list
global $mod;
$mod = [];
// Add route mechanismus
global $_URI;
$_URI = new URI();
// Add modules
foreach (glob('mod/*/module.inc.php') as $key =>
$val) {
 include_once($val);
}
// authentication on startup
global $_Auth;
$_Auth  = new Auth();
$_Auth->onstartup();
```

Supprimez *core/page.class.php*.

Modifiez *core/status.class.php* :

```
<?php
```

```
class Status {
 private array $mess;
 public function __construct () {
   $this->mess = [];
 }
 private function add (string $type, string $mess) :
void {
   if (!isset($this->mess[$type])) {
     $this->mess[$type] = [];
   }
   $this->mess[$type][] = $mess;
 }
 public function err (string $mess) : bool {
   $this->add('error',$mess);
   return false;
 }
 public function info (string $mess) : bool {
   $this->add('info',$mess);
   return true;
 }
 public function warn (string $mess) : bool {
   $this->add('warn',$mess);
   return true;
 }
 public function html () : string {
   reset($this->mess);
   $output = '';
   foreach ($this->mess as $key => $val) {
     if (count($val)) {
       foreach ($val as $key2 => $val2) {
         $output .= '
     <p class="'.$key.'">'.$val2.'</p>';
```

```
        }
      }
    }
    return $output;
  }
  public function text () : string {
    reset($this->mess);
    $output = '';
    foreach ($this->mess as $key => $val) {
      if (count($val)) {
        foreach ($val as $key2 => $val2) {
          $output .= "\n".$val2;
        }
      }
    }
    return $output;
  }
}
```

Presser par le temps (et la pause de midi...), Bob créer une révision.

```
git add -A
git commit -m "Modification de const.inc.php et de
core/status.class.php"
```

Au retour de la pause, il continue ses changements.

Créez *core/tpl_html.class.php* :

```
<?php
class Tpl_HTML {
  public array $header;
  public string $html;
  public array $hook;
  public array $var;
  public array $menu;
  public array $js;
  public array $css;
  public array $obj;
  public function __construct() {
    $this->header = [];
    $this->html = '';
    $this->hook = [];
    $this->var = [];
    $this->menu = [];
    $this->js = [];
    $this->css = [];
    $this->obj = [];
  }
  public function discover () : void {
    // retrieve hooks
    preg_match_all('/{{{(.*?)}}}/u',$this->html,
$hook);
    foreach ($hook[1] as $key => $val) {
      $this->hook[$val] = [];
    }
    // retrieve vars
    preg_match_all('/{{(.*?)}}/u',$this->html,$var);
    foreach ($var[1] as $key => $val) {
      $this->var[$val] = '';
    }
  }
  public function add_header (string $header) : void
{
```

```
    $this->header[] = $header;
  }
  public function add (string $id, string $data, int
$priority) {
    if (!isset($this->hook[$id])) {
      throw new Exception('Hook id "'.$id.'" not
found !');
    }
    $this->hook[$id][] = [
      'data' => $data,
      'priority' => $priority
    ];
  }
  public function set (string $id, string $data) {
    $this->var[$id] = $data;
  }
  public function parse () : string {
    // HTML headers
    reset($this->header);
    array_walk($this->header, 'header');
    // hooks handle
    $html = $this->html;
    reset($this->hook);
    foreach ($this->hook as $key => $val) {
      $hook = $this->hook[$key];
      // sort blocks inside hooks
      usort($hook, function ($item1, $item2) {
        return $item1['priority']-
$item2['priority'];
      });
      // concatenate blocks
      $data = join('', array_column($hook, 'data'));
      // replace hook
      $html = str_replace('{{{'.$key.'}}}',$data,
$html);
    }
    // replace vars
    reset($this->var);
    foreach ($this->var as $key => $val) {
      $html = str_replace('{{'.$key.'}}',$val,
$html);
    }
    print($html);
    exit(0);
  }
}
```

Créez une révision :

```
git add .
git commit -m "Ajout de core/tpl_html.class.php"
git status
```

Et là, horreur : il se rend enfin compte qu'il est sur la branche **master** au lieu d'être sur la branche **feature/modules** !

Comment récupérer ses changements, et les appliquer à la bonne branche ?

Heureusement, tant que les branches restent locales et n'ont pas été envoyées à distance, la correction est rapide, mais elle dépend de la situation.

Ici, nous avons une branche **feature/modules** existante, sur laquelle nous voulons rattacher les deux dernières révisions faites sur master. Nous ne pouvons pas faire un `rebase` de la branche

feature/modules, car cela reviendrait de fait à garder les deux révisions erronées dans **master**. Or nous voulons à la fin de l'opération les éliminer de **master**.

La solution est le "picorage" de révision, ou `cherry-pick`.

Allez sur la branche master (normalement on y est déjà...) :

```
git c.. m..
```

Affichez le journal simplifié sur une ligne :

```
git log --oneline
```

Repérez le dernier numéro de révision (ici noté BBBBBBB), et l'avant-avant dernier (ici noté AAAAAAA).

Revenez sur la branche **feature/modules**.

Affichez le journal simplifié sur une ligne :

```
git log --oneline
```

« Picorez » les deux derniers changements sous la forme d'une liste de type de ... à ... (attention à bien laisser les deux points)

```
git cherry-pick AAAAAAA..BBBBBBB
```

Affichez le journal simplifié sur une ligne :

```
git log --oneline
```

Les deux révisions ont bien été rajoutées à la branche de développement.

Revenez sur la branche master :

```
git s.. m..
```

Affichez le journal simplifié sur une ligne :

```
git log --oneline
```

Et supprimez les deux dernières révisions en « dur » :

```
git reset --hard HEAD^^
```

Explication : **HEAD** représente toujours la tête de la branche active. **HEAD^** le premier ancêtre en ligne directe. **HEAD^^** le second, etc.

La commande `git reset` accepte trois options :

➔ L'option `--soft` revient en arrière en gardant les changements dans l'index, sans modifier le dossier de travail.

➔ L'option `--mixed` revient en arrière en remettant l'index à zéro, sans modifier le dossier de travail. Cet l'option par défaut de la commande `git reset`.

➔ L'option `--hard` est la plus dangereuse : elle revient en arrière en supprimant réellement et définitivement les révisions intermédiaires et en modifiant donc le dossier de travail.

N.B. : Les deux premières options sont notamment utiles quand on veut réécrire et simplifier le journal en supprimant des révisions (et surtout des commentaires) jugés inutiles ou ennuyeux.

Affichez le journal simplifié sur une ligne :

```
git log --oneline
```

Et retournez sur la branche **feature/modules**.

Bob peut maintenant continuer son travail.

Modifiez *core/uri.class.php* :

```
<?php
class URI {
 private array $uri;
 public function __construct () {
   $this->uri = [];
 }
 public function register (mixed $parentpath, string
$path, string $title, string $namespace, string
$class, string $func) : void {
   if (!isset($this->uri[$path])) {
     $this->uri[$path] = [
       'parentpath' => $parentpath,
       'title' => $title,
       'namespace' => $namespace,
       'class' => $class,
       'func' => $func,
     ];
   }
 }
 public function unregister (string $path) {
   if (isset($this->uri[$path])) {
     unset($this->uri[$path]);
   }
 }
 public function call (string $path) : mixed {
   if (!isset($this->uri[$path])) {
     throw new Exception('Path '.$path .' not found
!');
   }
   $uri = $this->uri[$path];
   $n = $uri['namespace'];
   $c = $uri['class'];
   $f = $uri['func'];
   $nc = "$n\\$c";
   return (new $nc)->$f();
 }
 public function handle () : mixed {
   global $args;
   if (!isset($args['path'])) {
     $args['path'] = '/';
   }
   $data = $this->call($args['path']);
   if (isset($args['json'])) {

 header('Content-type:application/json;charset=utf-
8');
     die(json_encode($data)); // hash
   } else {
     include_once(__DIR__.'/../tpl/'.
$_SESSION['auth/usr/tpl'].'/tpl.class.php');
     global $_Tpl, $mod;
     $_Tpl = new tpl\default\Tpl();
     $_Tpl->init();
     reset($mod);
     foreach ($mod as $key => $val) {
       $func = $key.'\on';
       if (function_exists($func)) {
         $func('render');
       }
     }
     $_Tpl->merge($data);
```

```
      $_Tpl->parse();
    }
  }
}
```

Ajoutez le fichier à l'index :

```
git a.. .
```

Et créez une révision :

```
git c.. -m "Modification de core/uri.class.php"
```

Modifiez *mod/fr.storcka/module.inc.php* :

```
<?php
$_Mod = new Mod();
$_Mod->register('Page + Contact v1.0',__DIR__,'fr\
storcka');
include_once(__DIR__.'/page.class.php');
include_once(__DIR__.'/contact.class.php');
include_once(__DIR__.'/slot.inc.php');
// Page (always on by default)
$_URI->register(null, '/','Accueil','fr\
storcka','Page','handle');
$_URI->register(null, '/index.html','Accueil','fr\
storcka','Page','handle');
$_URI->register(null, '/index.php','Accueil','fr\
storcka','Page','handle');
// Enable contact
$_Mod->enable();
```

Ajoutez le fichier à l'index :

```
git a.. .
```

Et créez une révision :

```
git c.. -m "Modification de
mod/fr.storcka/module.inc.php"
```

Créez *mod/fr.storcka/slot.inc.php* :

```
<?php
namespace fr\storcka;
function on (string $action) {
 global $_URI;
 switch ($action) {
   case 'enabled':
   // Contact
     $_URI->register('/','/contact','Contact','fr\
storcka','Contact','form');

 $_URI->register('/','/contact/post','Contact','fr\
storcka','Contact','handle');
     break;
   case 'disabled':
     // Contact
     $_URI->unregister('/contact');
     $_URI->unregister('/contact/post');
     break;
   case 'render':
     global $mod, $_Tpl;
     $_Tpl->obj['menutop']->add('/','Accueil',10);
     if ($mod['fr\storcka']->enabled) {
```

```
$_Tpl->obj['menutop']->add('/contact','Contact',20);
        $_Tpl->obj['jstop']->add('contact.js',30);
      }
      break;
    default:
      // code...
      break;
  }
}
```

Ajoutez le fichier à l'index :

```
git a.. .
```

Et créez une révision :

```
git c.. -m "Création de mod/fr.storcka/slot.inc.php"
```

Et enfin modifiez *index.php* :

```
<?php
include('core/const.inc.php');
try {
  $data = $_URI->handle();
} catch (Exception $e) {
  if (ISCLI) {
    die($e->getMessage());
  }
  #mail('webmaster@lacigogne.net','Erreur sur le
site',$e->getMessage());
  $_Status->err('Désolé mais une erreur est survenue.
L\'équipe a été prévenue. Merci de revenir plus
tard.');
  die($_Status->text());
}
```

Ajoutez le fichier à l'index :

```
git a.. .
```

Et créez une révision :

```
git c.. -m "Modification de index.php"
```

Bob décide d'afficher le journal de sa branche, et s'aperçoit au final qu'il y a beaucoup trop d'informations pour rien. Il décide donc de simplifier son journal.

Affichez le journal simplifié sur une ligne :

```
git log --oneline
```

Supprimez les 7 derniers niveaux de révisions en conservant les fichiers modifiés, en gardant les modifications dans l'index (attention au caractère ~).

```
git s..
git reset HEAD~7 --soft
git s..
```

Affichez le journal simplifié sur une ligne :

```
git log --oneline
```

Vérifiez que les modifications sont bien là, et déjà prêtes à être intégrées dans une nouvelle révision :

```
git status
```

Et créez une nouvelle révision :

```
git c.. -m "Intégration des modules et mises à jour
de index.php, core/ et tpl/fr.stocka/"
```

Affichez le journal simplifié sur une ligne :

```
git log --oneline
```

Pour pouvoir tester son code, Bob ramène la branche **dev** mise à jour par Alice.

```
git s.. d..
git p.. o.. d..
```

Il retourne sur sa branche de développement :

```
git s.. f..
```

Et plutôt que d'utiliser un `merge`, il choisit ici un `rebase` de **dev** pour tester préalablement son code dans sa branche :

```
git r.. d..
```

GIT détecte notamment que la classe *core/page.class.php* a disparu en local, alors qu'elle existe dans **dev**. Il faut donc supprimer manuellement le fichier.

```
# windows
delete core/page.class.php
```

```
# GNU/Linux
rm core/page.class.php
```

C'est là où l'option `-A` (--all) de `git add` est indispensable, puisqu'elle va permettre d'indiquer à GIT qu'il faudra également supprimer le fichier sur le dépôt distant.

```
git add -A
git s..
git rebase --continue
```

Bob a maintenant tous les fichiers pour tester son code :

```
php index.php
```

Après vérifications, il décide de fusionner sa branche en mode avance rapide :

```
git s.. d..
git s..
git m.. f..
git s..
git p.. o.. d..
```

G) Conclusion

Si le projet ici proposé n'est pas finalisé (en fait, j'ai un code beaucoup plus avancé, mais à partir de là, c'est payant...), il vous aura permis de découvrir les principales commandes de GIT, et surtout les problèmes « classiques » du travail en groupe.

Il existe bien d'autres commandes (`man git` sous GNU/Linux) qui couvrent en fait tous les cas de figure rencontrés en développement, mais vous aurez compris que si GIT vous permet

de revenir en arrière, de remiser votre code, de picorer un code existant, de fusionner dans les deux sens vos modifications avec des dépôts distants, c'est toujours au développeur final de gérer les conflits de code. GIT ne vous fera pas le café !

Tant que les erreurs sont locales, ce qui arrive tôt ou tard quand on débute, ce n'est pas grave et on peut souvent s'en sortir assez facilement.

Par contre si vous réécrivez les journaux d'une branche distante commune, que vous l'effacez, que vous y postez des révisions intermédiaires qui n'intéressent personne et viennent polluer le fil, ça peut devenir gênant dans un groupe, et plus délicat à réparer.

Tout OS confondu, il existe un tas d'outils graphiques non abordés ici, le but étant vraiment de travailler en bas niveau pour bien comprendre ce qui se passe.

On ne va pas non plus se mentir : GIT a vraiment été pensé pour fonctionner en ligne de commande, et la plupart des outils graphiques n'apportent en fait pas grand chose face à une console GNU/Linux bien configurée, et quelques alias de commandes bien choisis...

Sinon l'idée générale de l'outil n'est pas de faire une révision à chaque fichier modifié, comme on le voit trop souvent dans des vidéos web, mais de faire une révision quand vous estimez avoir atteint une certaine stabilité de votre code.

Il faut se mettre 2s dans la peau du chef de projet qui doit savoir sur quoi vous travaillez et où vous en êtes dans la tâche qui vous est assignée.

Ainsi vouloir à tout prix finir sa fonctionnalité avant de faire une révision n'est pas une bonne stratégie : parfois, on peut se tromper de voie, et avoir des « révisions de secours » pas trop anciennes, permet souvent d'éviter les catastrophes...

L'art du développeur GIT est donc de trouver ce délicat équilibre entre ce qu'il faut fournir au groupe pour travailler de manière efficace et cohérente sur le projet, et ce qu'il veut garder pour lui en local.

Sur ce je vous souhaite une bonne et longue utilisation de GIT, et plein de succès dans vos nombreux programmes !

H) Annexe

H.1) Question en fin de cours

Un étudiant a demandé pourquoi il avait le comportement suivant : dans un nouveau dossier de développement, il avait fait un `git init` . puis tout de suite après il avait créé une branche sur laquelle il se mettait. Il avait ensuite importer des fichiers dans son dossier, mais après avoir fait un `git add/commit`, la branche `master` affichait encore ses fichiers copiés. En fait, ce comportement est normal : il faut faire une première révision sur **master** avant de commencer à travailler, car au départ, GIT n'a aucun historique de branche donc aucun différentiel… Tout fichier que vous créez/copiez appartient de fait à l'ensemble des branches présentes, d'où le comportement observé.

H.2) Archives au format TAR

```
# compression
tar cvf archive1.tar /home/serfa/toto
```

```
# visualisation
tar tvf archive1.tar
# extraction
tar xvf archive1.tar -C DOSSIER_DESTINATION
```

H.3) Archives au format TGZ

```
# tgz = tar + gzip
tar zcvf archive1.tgz /home/serfa/toto
tar ztvf archive1.tgz
tar zxvf archive1.tgz -C DOSSIER_DESTINATION
```

H.4) Astuces diverses sous GNU/Linux

➔ Autocomplétion des commandes / dossiers / fichiers : touche **TAB**

➔ Effacement de l'écran : **CTRL-l** `(ou clear)`

➔ Redonner les bons droits sur le **$HOME** :

```
chown -R serfa:serfa /home/serfa
```

➔ Passage en administrateur :

```
sudo -s ou sudo bash
```

➔ Quitter le mode administrateur :

```
CTRL+d (ou exit)
```

➔ Effacer un fichier :

```
rm NOM_FICHIER
```

➔ Effacer un dossier et son contenu :

```
rm -Rf NOM_DOSSIER
```

➔ Savoir dans quel dossier on est :

```
pwd (pour path working directory)
```

➔ Dans un fichier PHP, on peut mêler du code PHP délimité avec `<?php code PHP ?>` avec du code HTML - exemple :

```
<?php code PHP ?><p>CODE HTML</p><?php code PHP ?>
```

H.5) Astuces VIM

vim = éditeur avancé en console sur lequel se base de nombreux autres outils GNU/Linux. Il n'est généralement pas installé, ou pas complet (dépendances incomplètes), sur les distributions standards. Il vaut donc mieux toujours faire un `apt install vim` pour commencer...

VIM travaille en deux modes : insertion ou commande.

```
i, o, O : passage en mode insertion
ESC : passage en mode commande
```

En mode commande :

```
gg=G : réindentation automatique du code source
:wq : force write + quit
:q! : quitter sans modifier
```

```
SHIFT+z + SHIFT+z : quitter en sauvegardant
j : join - joindre des lignes
yy : mémoriser la ligne courante
5yy : mémoriser 5 lignes dont la ligne courante
p : ajouter la ligne courante après la ligne
actuelle
dd : effacer la ligne courante
3dd : effacer 3 lignes dont la courante
:%d : effacer tout le fichier
:4,$ : effacer de la ligne 4 jusqu'à la fin du
fichier
:32 : aller sur la ligne 32
/toto : rechercher la chaîne toto
:s/toto/tata : rechercher et remplacer la chaîne
toto en tata
:w nomfichier.txt : enregistrer le document courant
dans nomfichier.txt
:n : aller au document suivant si plusieurs
documents ont été fournis en paramètres
```

Bien entendu, ce n'est la qu'un micro-aperçu de ce que permet de faire l'outil, mais pour débuter, c'est déjà suffisant.

H.6) Retélécharger le cours / TP

```
wget
https://alsatux.com/doc/pub/serfa/git/formation.pdf
wget https://alsatux.com/doc/pub/serfa/git/cours.pdf
```

H.7) Autres tutoriaux Alsatux

J'ai créé de nombreux autres tutoriels destinés aux étudiants découvrant GNU/Linux que je vous invite bien entendu à découvrir et travailler de votre côté !

Cf. https://alsatux.com/UHA.