# Consolidation — Gérer les accès d'une équipe de développement en entreprise

## Contexte

Tu es administratrice système chez **WebForge**, une agence qui développe des sites web.

L'équipe vient de démarrer un projet Laravel nommé **boutique-en-ligne**, hébergé sur un serveur Linux que tu gères. Le projet doit être livrable, sécurisé, et accessible aux bonnes personnes.

Voici l'équipe telle qu'elle va exister sur ce projet :

| Rôle | Personne | Statut |
|---|---|---|
| Administrateur système / toi | ton compte habituel | Accès total, gère le serveur, ne code pas sur le projet |
| Développeuse interne (lead dev) | **sophie** | Doit pouvoir tout faire sur le code du projet |
| Développeur interne | **marc** | Doit pouvoir tout faire sur le code du projet |
| Freelance externe | **paul** | Doit pouvoir travailler sur le code, **sauf** sur les fichiers sensibles (configuration, secrets) |
| Serveur web | **www-data** | Doit pouvoir écrire uniquement dans certains dossiers précis du projet, sans avoir la main sur le reste du code |

Ce document reprend, dans un seul scénario continu et réaliste, tout ce que tu as manipulé dans les séances précédentes : chemins, permissions, utilisateurs et groupes, `setgid` et sticky bit. Comme d'habitude, aucune commande n'est donnée : à toi de la trouver.

## Le rapport à me transmettre

Utilise la même méthode que pour les séances précédentes.

## Phase 1 — Constituer l'équipe

### 1.1 — Le groupe projet

Crée un groupe nommé `webforge_boutique`, qui représentera "toutes les personnes autorisées à travailler sur le code de ce projet".

### 1.2 — Les comptes de l'équipe

Crée les comptes `sophie`, `marc` et `paul`, chacun avec `webforge_boutique` comme groupe **secondaire**. Donne un mot de passe à chacun.

### 1.3 — Vérification d'usage

Vérifie pour chacun des trois comptes que le groupe a bien été pris en compte.

## Phase 2 — Installer le projet correctement

### 2.1 — Un vrai projet, pas une simulation

Installe une **application Laravel neuve** dans `/var/www/html`. Laravel viendra avec sa propre arborescence par défaut (`app/`, `config/`, `storage/`, `public/`, etc.).

Une fois le projet installé, assure-toi que le fichier `.env` existe bien à la racine du projet et contient au moins une ligne simulant un secret d'entreprise.

### 2.2 — Propriété et droits de base

Fais en sorte que l'ensemble du projet appartienne au groupe `webforge_boutique`, avec des droits qui permettent à ce groupe de tout lire et tout modifier, et qui interdisent tout accès aux utilisateurs extérieurs à ce groupe.

### 2.3 — L'héritage automatique du groupe

Configure le dossier du projet pour que **tout fichier créé par n'importe qui à l'intérieur**, aujourd'hui ou plus tard, hérite automatiquement du groupe `webforge_boutique`, sans que chaque développeur ait à y penser à chaque création de fichier.

### 2.4 — Vérification par la pratique

Bascule vers `marc`, crée un fichier `Controller.php` dans `app/`.

Bascule vers `sophie`, crée un fichier `Test.php` dans `app/`.

Bascule vers `paul`, crée un fichier `App.php` dans `app/`.

Reviens à ton compte et observe le résultat. Explique ce qui se passe et règle le problème.

## Phase 3 — Intégrer le serveur web sans tout lui donner

Dans un vrai déploiement Laravel, le processus du serveur web (ici représenté par `www-data`) a besoin d'écrire uniquement dans `storage/` (logs, fichiers temporaires, cache) — jamais dans le reste du code de l'application, pour limiter les dégâts si le site web est un jour compromis.

### 3.1 — Un accès limité, pas un accès total

Donne à `www-data` la capacité d'écrire dans `storage/` et uniquement ce dossier, et sans l'ajouter au groupe `webforge_boutique`.

### 3.2 — Vérification

Bascule vers `www-data` (ou simule son comportement), essaie d'écrire un fichier dans `storage/`, puis essaie d'en écrire un dans `app/`. Observe le résultat, explique ce qui se passe et règle le problème. Le premier doit réussir, le second doit échouer.

## Phase 4 — Le cas du freelance

Paul doit pouvoir travailler sur le code (`app/`, `public/`), mais **jamais** voir le contenu de `.env`, même s'il fait partie du groupe `webforge_boutique` qui, lui, a accès en lecture à tout le projet par défaut.

### 4.1 — Une exception fine, sans casser le reste

Restreins spécifiquement l'accès de `paul` au fichier `.env`, pour qu'il ne puisse ni le lire ni y écrire, **sans** changer les droits classiques du fichier (propriétaire/groupe/autres doivent rester identiques pour tout le monde d'autre), et **sans** le retirer du groupe `webforge_boutique`.

### 4.2 — Vérification croisée

Vérifie que `paul` ne peut plus accéder à `.env`, mais que `sophie` et `marc`, eux, le peuvent toujours normalement.

### 4.3 — Repérer la particularité à l'œil nu

Affiche les détails du fichier `.env` d'une façon qui te permette de repérer, rien qu'en un coup d'œil sur la ligne de droits, qu'un mécanisme de permission "non standard" est appliqué dessus. Confirme ensuite en détail avec l'outil dédié à ce mécanisme.

## Phase 5 — Un espace de dépôt temporaire partagé

L'équipe a besoin d'un dossier commun `depot_livrables/` où chacun peut déposer des fichiers à faire relire (captures d'écran, exports, fichiers de test) — accessible en écriture à tout le monde sur le serveur, mais où **personne ne doit pouvoir supprimer les fichiers déposés par quelqu'un d'autre**, y compris toi.

### 5.1 — Mise en place

Crée ce dossier dans `/srv`, avec les droits et le mécanisme qui correspondent à ce besoin exact.

### 5.2 — Vérification en conditions réelles

Fais déposer un fichier par `marc`. Bascule vers `paul` et essaie de le supprimer : ça doit échouer. Bascule vers ton propre compte (avec des droits élevés) et retente : que se passe-t-il, et pourquoi ce cas précis fonctionne quand même malgré le mécanisme mis en place ?