Retour au dossier de maintenance	
Sujets connexes	Cartographie fonctionnelle de LoGeAs Informations sur la version en cours Proc#145 : Procédure de Gestion des Versions Proc#168 - Chaîne de qualification et de livraison Proc#167 - Procédure de mise en ligne d'une version du client « lourd » Proc#169 - Cycle de release LoGeAs Web

Proc#170 - Guide pratique — Gestion des versions LoGeAs Web

Informations qualité

Suivi des modifications majeures		
Date	**Auteur**	**Modifications**
11 août 2026	Nicolas MARCHAND	Création du document
30 septembre 2026	Nicolas MARCHAND	Alignement sur la chaîne de qualification et de livraison (Proc#168) : le guide se limite au travail personnel, à l'intégration et à la bascule entre versions, et renvoie à Proc#168 pour la qualification, la beta, la recette et la release (suppression des anciennes étapes de publication et de la release sans beta) ; mise au format de la maîtrise documentaire
Suivi des approbations	Cartographie fonctionnelle – fiche de la procédure	
Objet	Ce guide décrit les opérations à effectuer pour gérer les versions de LoGeAs Web, depuis le développement individuel jusqu'à la publication d'une version destinée aux utilisateurs.	
Destinataires	**- Validation des modifications :** Gérant **- Approbation du document :** Équipe développement	

1. Objet

Ce guide décrit les opérations pratiques de gestion des versions de LoGeAs Web. Il s'adresse principalement aux développeurs et aux personnes chargées de préparer les publications.

Il complète les documents suivants :

- Proc#145 - Procédure de gestion des versions : règles générales de versionnage.
- Proc#168 - Chaîne de qualification et de livraison : **déroulé pas à pas de la qualification, de la beta, de la recette et de la release**, portes bloquantes, cas particuliers.
- Proc#169 - Cycle de release LoGeAs Web : fonctionnement technique des scripts.

Répartition : ce guide détaille le travail quotidien du développeur (branches, commits, intégration, bascule entre versions). Tout ce qui va de la qualification à la livraison est décrit dans Proc#168, et

uniquement là.

2. Environnement de travail

2.1 Outils

Les opérations sont réalisées depuis le poste de développement, avec :

- Git et l'accès aux dépôts distants ;
- Node.js et npm ;
- Angular CLI ;
- Claude Code, pour les commandes d'assistance du cycle ;
- les accès aux dépôts `logeas-web`, `logeas-lib` et `logeas-cartographie` ;
- un accès connecté à la cartographie fonctionnelle, pour valider les rapports qualité et faire la recette.

Les commandes `/fusion-dev`, `/qualifier-version`, `/release-beta`, `/release-client` et `/rapport-version-qualite` sont des commandes Claude Code. Elles ne se saisissent pas comme des commandes shell.

2.2 Répertoires

Répertoire	**Usage**
`D:\Developpements\Sources\NewInterfaces\logeas-web`	Dépôt principal de l'interface et des scripts de livraison
`D:\Developpements\Sources\NewInterfaces\logeas-lib`	Dépôt des bibliothèques partagées
`D:\Developpements\Sources\NewInterfaces\logeas-cartographie`	Dépôt de la cartographie fonctionnelle (écran « Rapports de version »)
`D:\Developpements\VersionPublic\LogeasWeb-Test\browser`	Résultat du build beta
`D:\Developpements\VersionPublic\LogeasWeb\browser`	Résultat du build de production
`D:\Developpements\VersionPublic\ExecutablesWindows`	Exécutables des composants utilisés pour les rapports de release

Les commandes Claude Code et les scripts se lancent depuis la racine du dépôt `logeas-web`.

3. Vue d'ensemble

Étape	Commande ou opération	Qui	Décrit dans
1. Travail personnel	Git, commits et sauvegardes	Développeur	ce guide, section 4
2. Intégration	`/fusion-dev`	Développeur	ce guide, section 5
3. Qualification	`scripts\qualifier-version.bat`	Développeur	Proc#168
4. Validation du rapport qualité	carto, écran « Rapports de version »	Valideur	Proc#168
5. Beta	`scripts\livrer-beta.bat`	Développeur	Proc#168
6. Déploiement sur le serveur de test	copie manuelle	Personne habilitée	Proc#168
7. Recette de la beta	carto, écran « Rapports de version »	Valideur	Proc#168
8. Release client	`scripts\livrer-release.bat`	Développeur	Proc#168
9. Déploiement en production	copie manuelle	Personne habilitée	Proc#168

4. Étape 1 — Travail personnel

4.1 Création de la branche

Dans le dépôt `logeas-web`, créer une branche personnelle à partir de la branche de version en cours.

Exemple :

```
git checkout 11.2.0
git pull
git checkout -b feature/nom-fonctionnalite
```

Le nom de la branche doit permettre d'identifier le travail effectué.

4.2 Développement et sauvegarde

Le développeur réalise ses modifications et les sauvegarde régulièrement.

Exemple :

```
git status
git add src/app/mon-composant
git commit -m "Ajout de la fonctionnalité"
git push -u origin feature/nom-fonctionnalite
```

Avant chaque commit, vérifier les fichiers sélectionnés.

Ne pas ajouter de fichiers contenant des secrets, des paramètres locaux ou des fichiers de compilation qui ne sont pas destinés au dépôt.

Les tests peuvent être exécutés pendant le développement selon les besoins. Cette étape constitue

une sauvegarde du travail personnel et non une validation de l'intégration.

4.3 Fin du travail personnel

Lorsque le travail est prêt à être intégré :

- vérifier les modifications ;
- sauvegarder les derniers commits ;
- s'assurer que la branche personnelle est disponible sur le dépôt distant ;
- lancer l'intégration.

La branche personnelle ne doit pas être fusionnée directement dans la branche commune en contournant la procédure d'intégration.

5. Étape 2 — Intégration

5.1 Préparation

L'intégration est réalisée avec la commande :

```
/fusion-dev
```

La commande se lance dans `logeas-web`. Elle identifie la branche de version commune et réalise la fusion de la branche personnelle.

Avant de poursuivre, vérifier que :

- le travail est prêt à être intégré ;
- la branche personnelle a été sauvegardée ;
- les éventuels conflits ont été examinés ;
- la branche cible correspond bien à la version en cours.

5.2 Contrôles

La procédure réalise les contrôles techniques après la fusion :

- compilation ;
- tests unitaires ;
- lint.

Le développeur vérifie le résultat global, notamment les avertissements et les échecs.

En cas de conflit concernant le fonctionnement métier, ne pas accepter une résolution automatique sans vérification.

En cas d'échec bloquant, la branche commune ne doit pas être publiée tant que le problème n'est pas traité.

5.3 Résultat attendu

L'intégration est terminée lorsque :

- la fusion est effectuée ;
- les contrôles prévus ont été réalisés ;
- les éventuels problèmes ont été résolus ;
- la branche commune a été publiée sur le dépôt distant.

Le commit de fusion permet de retrouver les changements intégrés.

6. De la qualification à la livraison

La qualification, la validation du rapport qualité, la beta, la recette, la release et les déploiements ne sont pas décrits ici : ils le sont pas à pas dans Proc#168 - Chaîne de qualification et de livraison, sections « Cheminements pas à pas » et « Les étapes en détail ».

À retenir une fois l'intégration terminée :

- la suite commence par `scripts\qualifier-version.bat`, après avoir **poussé** les branches de version de `logeas-web` et de `logeas-lib` ;
- **ne rien commiter ni pousser pendant la qualification**, ni entre une beta et sa release ;
- tout correctif après une beta repart d'une branche personnelle, de `/fusion-dev`, puis d'une nouvelle qualification ;
- il n'y a pas de release sans beta recettée conforme.

7. Cas particulier — Démarrer une nouvelle version

Lorsqu'une évolution nécessite de passer à une nouvelle version (Proc#168, cas 6), créer la nouvelle branche de version dans `logeas-web` et, si la bibliothèque évolue aussi, dans `logeas-lib`.

Exemple :

```
git checkout 11.2.0
git pull
git checkout -b 11.3.0
git push -u origin 11.3.0
```

La branche doit être **poussée** : une branche sans branche amont est refusée par la qualification.

L'ancienne branche est conservée avec son historique et ses tags. La nouvelle branche devient la cible des prochaines intégrations.

Informer l'équipe de ce changement afin que les développements en cours soient correctement

réorientés.

La version candidate est tirée du nom de la branche : la première beta de `11.3.0` sera `11.3.0.1`.

8. Cas particulier — Correctifs et évolutions en parallèle

Deux pistes de développement peuvent être maintenues simultanément :

Piste	Usage
Version en production	Correctifs de la version utilisée par les clients
Version suivante	Nouvelles fonctionnalités et évolutions

Les corrections apportées à la version de production ne sont pas automatiquement reportées sur la version suivante : il faut les y intégrer (Proc#168, cas 7).

8.1 Bascule entre versions

`logeas-web` utilise les bibliothèques de `logeas-lib` sous forme de paquets construits. Un simple changement de branche Git ne suffit pas à garantir que les dépendances correspondent à la version souhaitée.

Le script suivant bascule le poste sur une paire de branches web / lib, reconstruit la bibliothèque et la réinstalle :

```
scripts/switch-lib-version.ps1
```

Les profils sont définis dans :

```
scripts/switch-lib-profiles.json
```

Exemples :

```
.\scripts\switch-lib-version.ps1 -Profile hotfix
```

```
.\scripts\switch-lib-version.ps1 -Profile next
```

```
.\scripts\switch-lib-version.ps1 -LibBranch 11.3.0 -WebBranch 11.3.0
```

Avant d'utiliser cet outil, vérifier que les dépôts ne contiennent pas de modifications locales non sauvegardées.

Le script signale les correctifs d'une version antérieure pas encore reportés sur l'autre piste. Examiner ces écarts avant de poursuivre.

Les profils doivent être actualisés lorsqu'une nouvelle organisation de branches est mise en place.

9. Commandes et précautions

9.1 Commandes Claude Code

Commande	Utilisation
`/fusion-dev`	Intégrer le travail personnel à la branche de version
`/qualifier-version`	Vérifier les préconditions de la qualification, puis renvoyer vers `qualifier-version.bat`
`/release-beta`	Vérifier la porte beta, expliquer ce qui bloque, puis renvoyer vers `livrer-beta.bat`
`/release-client`	Vérifier la porte release, expliquer ce qui bloque, puis renvoyer vers `livrer-release.bat`
`/rapport-version-qualite`	Rédiger l'analyse du rapport qualité (appelée par la qualification)

Seule `/fusion-dev` modifie le dépôt. Les commandes `/qualifier-version`, `/release-beta` et `/release-client` **ne livrent rien elles-mêmes**.

9.2 Génération manuelle d'un build

Les commandes suivantes génèrent un build **hors chaîne** :

```
ng build --configuration "production,beta"
```

```
ng build --configuration production
```

Elles ne passent pas les portes et ne créent ni rapport, ni tag. `npm run build` et `npm run build:beta` refont en plus la mise à jour du numéro de version. Dans tous les cas, la version produite est affichée « Version non qualifiée ».

Elles ne doivent pas être utilisées pour une livraison normale. Leur usage en urgence est décrit dans Proc#168, cas 8.

9.3 Précautions générales

Avant toute opération :

- vérifier le dépôt et la branche active ;
- vérifier l'état des modifications locales ;
- ne jamais utiliser `git push –force` sur une branche commune ;
- ne pas supprimer ou réécrire les tags d'une version publiée ;
- **ne jamais contourner une porte** : si le blocage est injustifié, c'est un défaut des scripts, à signaler et corriger ;
- ne pas confondre génération d'un build et déploiement d'une version.

En cas de doute sur la branche cible, le numéro de version ou la validité d'une publication, suspendre l'opération et demander une vérification.

10. Aide-mémoire Git et SourceTree

Les scripts exécutent automatiquement les opérations Git prévues. Le tableau ci-dessous sert au dépannage et aux vérifications manuelles.

Commande Git	**Équivalent SourceTree**
`git fetch`	Bouton Fetch
`git checkout <branche>`	Double-clic sur la branche
`git pull`	Bouton Pull
`git status`	Onglet File status
`git add <fichiers>`	Sélection des fichiers dans File status
`git commit`	Bouton Commit
`git merge –no-ff <branche>`	Merge avec création d'un commit même en cas de fast-forward
`git tag -a`	Création d'un tag annoté sur le commit
`git push`	Bouton Push, avec sélection des tags à publier
`git log <a>..<b>`	Consultation de l'historique entre deux références

Un tag créé localement mais non publié ne constitue pas une preuve de publication partagée.

11. Révision du guide

Ce guide doit être actualisé lorsque les commandes, les scripts, les branches de version ou les modalités de publication évoluent.

Les modifications doivent rester cohérentes avec Proc#168 et Proc#169.

From:
https://wiki-logeas.fr/certif/ - **dokuwiki-certif**

Permanent link:
https://wiki-logeas.fr/certif/doku.php?id=certif:procedure:develop:gestionversionpratique

Last update: **2026/09/30 10:38**