Retour au dossier de maintenance

| | |
|---|---|
| **Sujets connexes** | Cartographie fonctionnelle de LoGeAs<br>Informations sur la version en cours Proc#145 : Procédure de Gestion des Versions<br>Proc#168 - Chaîne de qualification et de livraison<br>Proc#167 - Procédure de mise en ligne d'une version du client « lourd »<br>Proc#169 - Cycle de release LoGeAs Web<br>Proc#170 - Guide pratique — Gestion des versions LoGeAs Web |

# Guide pratique — Gestion des versions LoGeAs Web

## Informations qualité

| | |
|---|---|
| **Suivi des modifications majeures** | septembre 2026 - Nicolas Marchand - Création du document |
| **Suivi des approbations** | https://cartographie-fonctionnelle.logeas-web.fr/UniversDoc |
| **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<br>**- Approbation du document** : Equipe dev |

## 1. 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.

Il s'adresse principalement aux développeurs et aux personnes chargées de préparer les publications.

Il complète les documents suivants :

- Procédure de gestion des versions : règles générales de versionnage.
- Chaîne de qualification et de livraison : responsabilités, contrôles et validations.
- Cycle de release LoGeAs Web : fonctionnement technique des procédures automatisées.

Ce guide décrit les opérations pratiques. En cas de différence entre une commande et le fonctionnement réel du script, le script et la procédure technique doivent être vérifiés avant toute publication.

## 2. Environnement de travail

### 2.1 Outils

Les opérations de release 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 automatisées du cycle ;
- les accès aux dépôts `logeas-web`, `logeas-lib` et, pour le rapport qualité, `logeas-cartographie`.

Les commandes `/fusion-dev`, `/release-beta`, `/release-client` et `/rapport-tests-qualite` sont des commandes Claude Code. Elles ne doivent pas être saisies comme des commandes shell classiques.

### 2.2 Répertoires

| **Répertoire** | **Usage** |
|---|---|
| `D:\Developpements\Sources\NewInterfaces\logeas-web` | Dépôt principal de l'interface |
| `D:\Developpements\Sources\NewInterfaces\logeas-lib` | Dépôt des bibliothèques partagées |
| `D:\Developpements\Sources\NewInterfaces\logeas-cartographie` | Dépôt des rapports et de la cartographie |
| `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 doivent être lancées depuis la racine du dépôt concerné.

## 3. Vue d'ensemble

| **Étape** | **Commande ou opération** | **Responsable de la décision** |
|---|---|---|
| 1. Travail personnel | Git, commits et sauvegardes | Développeur |
| 2. Intégration | `/fusion-dev` | Développeur |
| 3. Publication beta | `/release-beta` | Chef de projet ou développeur senior |
| 4. Validation beta | Tests et recette | Responsable de la validation |
| 5. Publication client | `/release-client` | Chef de projet |
| 6. Déploiement | Copie vers le serveur | Personne habilitée |

Une release directe, sans beta préalable, est possible lorsqu'elle est justifiée et que les contrôles applicables sont réalisés.

# 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 11.0.25
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 suffisamment avancé et prêt à être intégré :

- vérifier les modifications ;
- sauvegarder les derniers commits ;
- s'assurer que la branche personnelle est disponible sur le dépôt distant ;
- demander ou 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 doit être lancée 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 doit vérifier le résultat global et notamment les éventuels avertissements ou 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 ou traités selon la procédure applicable ;
- la branche commune a été publiée sur le dépôt distant.

Le commit de fusion permet de retrouver les changements intégrés.

# 6. Étape 3 — Publication d'une beta

## 6.1 Conditions préalables

La publication beta est décidée lorsque la branche de version est considérée comme suffisamment stable pour être testée.

Avant de lancer la procédure, vérifier :

- que les développements à intégrer sont bien présents ;
- que les intégrations nécessaires ont été réalisées ;
- que la branche commune est propre et à jour ;
- que les contrôles d'intégration sont satisfaisants.

## 6.2 Génération

Dans `logeas-web`, sur la branche de version, lancer :

```
/release-beta
```

La commande réalise automatiquement les opérations prévues dans la procédure technique, notamment :

- vérification du dépôt ;
- calcul du numéro beta ;
- génération des informations de version ;
- compilation de l'application ;
- génération du rapport de release ;
- création du commit de préparation ;
- création et publication du tag Git.

Le build est produit dans :

```
D:\Developpements\VersionPublic\LogeasWeb-Test\browser\
```

## 6.3 Vérifications après génération

Après l'exécution, vérifier :

- que la commande s'est terminée sans erreur bloquante ;
- que le numéro beta correspond au cycle attendu ;
- que le tag beta a été créé et publié ;
- que le build existe dans le répertoire de test ;
- que le rapport de release est accessible ;
- que les éventuelles anomalies signalées sont prises en compte.

Le numéro beta utilise le format `X.Y.Z.W_beta`.

### 6.4 Déploiement sur le serveur de test

Le transfert du build vers le serveur de test reste manuel.

La personne habilitée doit :

- récupérer le contenu du répertoire de génération beta ;
- le transférer vers le serveur de test par le moyen d'accès autorisé ;
- vérifier que les fichiers ont été copiés dans le répertoire prévu ;
- vérifier le fonctionnement de la version déployée.

La beta peut alors être soumise aux tests et à la recette prévus par la chaîne de qualification et de livraison.

## 7. Étape 4 — Validation de la beta

La validation de la beta est une étape distincte de sa génération.

Le responsable de la validation vérifie les fonctionnalités concernées et les résultats des tests prévus.

Les anomalies constatées doivent être enregistrées et traitées.

Lorsque des corrections sont nécessaires :

- le développeur réalise les modifications sur une branche personnelle ;
- les corrections sont intégrées par `/fusion-dev` ;
- une nouvelle beta peut être générée avec `/release-beta` ;
- les tests nécessaires sont renouvelés.

Une beta ne doit pas être considérée comme validée uniquement parce que sa compilation a réussi.

La décision de poursuivre vers une release client est prise par le responsable habilité, conformément à la chaîne de qualification et de livraison.

## 8. Étape 5 — Publication client

### 8.1 Conditions préalables

Avant la publication client, vérifier :

- que la version à livrer est identifiée ;
- que les validations requises sont réalisées ;
- que les anomalies bloquantes sont traitées ;
- que la branche de version est propre et à jour ;
- que les modifications apportées depuis la dernière beta ont été examinées, lorsqu'une beta existe.

Si des changements fonctionnels importants ont été ajoutés après la beta, une nouvelle phase de validation peut être nécessaire.

## 8.2 Génération

Dans `logeas-web`, sur la branche de version, lancer :

```
/release-client
```

La commande réalise les opérations de préparation de la release :

- vérification du dépôt ;
- calcul du numéro de release ;
- génération du build de production ;
- génération du rapport de release ;
- création du commit de préparation ;
- création et publication du tag de release ;
- préparation du rapport qualité NF dans le dépôt de cartographie.

Le build est généré dans :

```
D:\Developpements\VersionPublic\LogeasWeb\browser\
```

Le numéro de release utilise le format `X.Y.Z_release` pour le tag.

## 8.3 Vérifications après génération

Après l'exécution, vérifier :

- que la génération s'est terminée sans erreur bloquante ;
- que le numéro de version est cohérent avec le cycle ;
- que le tag de release est correct ;
- que le build de production est présent dans le répertoire attendu ;
- que le rapport de release est disponible ;
- que le rapport qualité NF a été régénéré et peut être relu.

Le rapport qualité NF doit être relu avant son intégration dans `logeas-cartographie`. Sa validation et son commit ne doivent pas être considérés comme automatiques.

## 8.4 Release sans beta préalable

Une release peut exceptionnellement être générée sans beta préalable.

Cette possibilité ne dispense pas des contrôles techniques et de la validation nécessaires à la livraison.

La décision et sa justification doivent être conservées dans les éléments de suivi de la livraison, selon les modalités prévues par la chaîne de qualification.

# 9. Étape 6 — Déploiement en production

Le déploiement est réalisé après la décision de livraison.

Il reste manuel et nécessite l'accès au serveur de production.

La personne habilitée doit :

- récupérer le contenu du répertoire de production ;
- transférer les fichiers vers le serveur concerné ;
- vérifier la bonne copie des fichiers ;
- contrôler l'accès à l'application après déploiement ;
- signaler toute anomalie constatée.

Le tag Git et le rapport de release permettent d'identifier la version source du déploiement.

Les opérations de déploiement et les éventuels contrôles complémentaires sont documentés dans les procédures d'exploitation applicables.

# 10. Cas particulier — Changement de version mineure

Lorsqu'une évolution nécessite de passer à une nouvelle version mineure en cours de cycle, une nouvelle branche de version peut être créée.

Exemple :

```
git checkout 11.0.25
git pull
git checkout -b 11.1.0
git push -u origin 11.1.0
```

L'ancienne branche est conservée avec son historique et ses tags.

La nouvelle branche devient la cible des prochaines intégrations.

Il faut informer l'équipe de ce changement afin que les développements en cours soient correctement réorientés.

Le calcul automatique de la version tient compte du numéro de la nouvelle branche lors de la prochaine génération beta.

Les règles détaillées de numérotation sont décrites dans la procédure technique du cycle de release.

# 11. Cas particulier — Correctifs et évolutions en parallèle

Deux pistes de développement peuvent être maintenues simultanément :

- une piste de correctifs destinée à la version actuellement utilisée par les clients ;
- une piste d'évolution destinée à une prochaine version.

Exemple :

| **Piste** | **Usage** |
|---|---|
| Version en production | Correctifs urgents et corrections nécessaires |
| 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 donc vérifier que les correctifs nécessaires sont également intégrés à la branche d'évolution.

## 11.1 Changement de version des bibliothèques

`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 permet d'automatiser le changement de configuration :

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

Les profils sont définis dans :

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

Exemples de commandes :

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

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

Les profils déterminent les branches à utiliser pour `logeas-web` et `logeas-lib`.

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

Le script peut également signaler des commits présents sur une piste et absents de l'autre. Il faut examiner ces écarts avant de poursuivre.

Après un changement de version, vérifier que les paquets ont été reconstruits et que les dépendances installées correspondent à la branche sélectionnée.

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

# 12. Commandes et précautions

## 12.1 Commandes de publication

| Commande | Utilisation |
|---|---|
| `/fusion-dev` | Intégrer le travail personnel à la branche commune |
| `/release-beta` | Générer et publier une beta |
| `/release-client` | Générer et publier une release client |
| `/rapport-tests-qualite` | Régénérer le rapport qualité NF |

Ces commandes sont exécutées dans Claude Code, depuis le dépôt approprié.

## 12.2 Génération manuelle d'un build

Les commandes Angular suivantes permettent de générer un build sans réaliser l'ensemble du processus de publication :

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

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

Ces commandes ne remplacent pas les procédures de release.

Elles ne réalisent pas nécessairement le calcul de version, la génération du rapport, la création des tags ou les autres opérations de traçabilité.

Elles ne doivent donc pas être utilisées pour produire directement une livraison officielle destinée aux utilisateurs.

## 12.3 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 pas contourner les contrôles bloquants ;
- 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.

# 13. Aide-mémoire Git et SourceTree

Les commandes de release exécutent automatiquement les opérations Git prévues. Le tableau ci-dessous sert principalement 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 |

Lors d'une publication manuelle, vérifier que le tag est bien envoyé au dépôt distant. Un tag créé localement mais non publié ne constitue pas une preuve de publication partagée.

# 14. Traçabilité et conservation

Pour chaque version publiée, les éléments suivants doivent pouvoir être retrouvés :

- la branche de version concernée ;
- les commits intégrés ;
- le commit de préparation de version ;
- le tag beta ou release ;
- les informations de version embarquées ;
- le rapport de release ;
- les résultats de qualification et de recette ;
- les éléments de validation et de déploiement applicables.

Les rapports de release sont accessibles depuis l'application et les outils de suivi prévus à cet effet.

Le suivi des versions, des tests et des validations est assuré par les outils de cartographie fonctionnelle et de certification.

Les règles de conservation et d'archivage des preuves sont définies dans la chaîne de qualification et de livraison.

# 15. 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 la procédure technique de release et la chaîne de qualification et de livraison.

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

Permanent link:
**https://wiki-logeas.fr/certif/doku.php?id=certif:procedure:develop:gestionversionpratique&rev=1790660716**

Last update: **2026/09/29 07:45**