

 [Retour au Dossier Organisationnel](#)

 **Sujets connexes**

[Cartographie fonctionnelle de LoGeAs](#)
[Procédure de Gestion des Versions](#)

Guide pratique — Cycle de release, étape par étape

Document complémentaire à [Procédure de Gestion des Versions](#) — celui-ci répond à la question « qu'est-ce que je fais, concrètement, à chaque étape, qui le fait, et où ? ». Destiné au développeur et au chef de projet, pas à un usage certification.

Où se lancent ces commandes

`/fusion-dev`, `/release-beta`, `/release-client` et `/rapport-tests-qualite` sont des **commandes Claude Code** (l'assistant IA installé sur le poste de développement), pas des commandes shell classiques. Elles se tapent directement dans l'interface Claude Code (CLI ou extension), pas dans un terminal Git Bash / PowerShell nu.

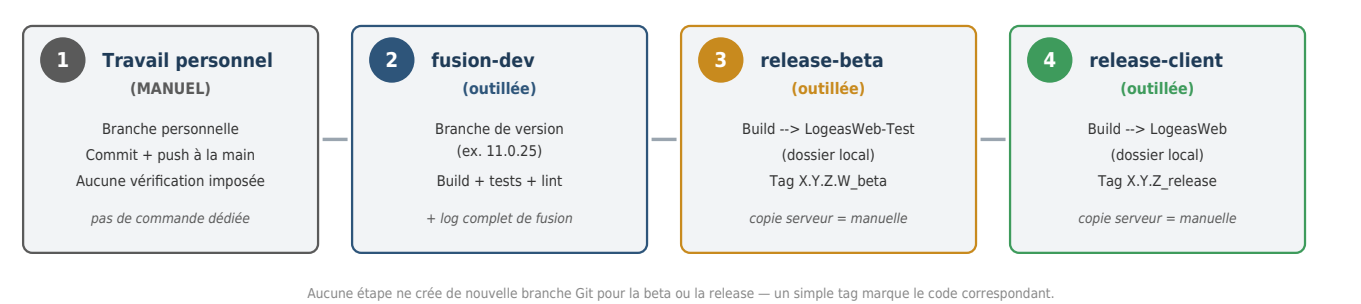
- **Poste concerné** : celui qui a un checkout local de ```logeas-web``` **et** accès au dossier ```D:\Developpements\VersionPublic\ExecutablesWindows\``` (pour les versions des composants Galaxie LoGeAs dans le rapport de release).
- **Dossier de travail** : toujours la racine du dépôt ```logeas-web``` (ou ```logeas-cartographie``` pour la dernière étape de `/release-client`).
- **Branche active au moment de lancer la commande** :
 - `/fusion-dev` : peu importe la branche de départ, la commande fait elle-même le checkout vers la branche de version.
 - `/release-beta` et `/release-client` : doivent être lancées **sur la branche de version** (ex. `11.0.25`), pas sur une branche personnelle.

Qui fait quoi

Étape	Qui décide de lancer	Qui exécute
1. Travail personnel	Développeur (son propre rythme)	Développeur
2. <code>/fusion-dev</code>	Développeur (son travail est prêt)	Développeur
3. <code>/release-beta</code>	Chef de projet ou développeur senior (juge la branche stable)	Développeur ou chef de projet — l'un ou l'autre peut techniquement lancer la commande
4. <code>/release-client</code>	Chef de projet (validation métier que la beta est bonne à livrer)	Développeur ou chef de projet
Copie manuelle vers le serveur (test ou client)	—	Personne ayant accès au bureau à distance du serveur concerné

En résumé : le développeur porte les étapes 1 et 2 seul. À partir de l'étape 3, c'est une décision d'équipe/de chef de projet de passer à l'étape suivante — l'exécution technique de la commande peut rester déléguée au développeur, mais le feu vert est côté chef de projet, en particulier pour l'étape 4 (on livre au client).

Vue d'ensemble



travail potentiellement inachevé, pas encore un contrôle qualité.

À vérifier soi-même avant de committer :

- Rien de sensible n'est ajouté (secrets, fichiers de config locale, `dist/`, `coverage/`).
- Si le changement touche en réalité un comportement partagé (```DGService```, composants génériques), le code doit vivre dans ```logeas-lib```, pas ici.

Étape 2 — Intégration (``/fusion-dev``)

Déclencheur : le travail sur la branche personnelle est prêt à rejoindre le tronc commun.

Qui : le développeur qui a fait le travail (ou toute personne de l'équipe).

Où : Claude Code, dans ```logeas-web```.

Ce que la commande fait, dans l'ordre :

1. `git fetch`
2. `git checkout <branche-de-version>` puis `git pull`
3. `git log <branche-de-version>..<branche-perso> --pretty=...` — récupère la liste complète des commits qui vont être intégrés
4. `git merge --no-ff <branche-perso>` — le message de fusion embarque cette liste de commits, pas juste un titre générique
5. `ng build` — vérifie que ça compile
6. `npm test` — tests unitaires (Jest)
7. `npm run lint` — seulement les **nouvelles** erreurs introduites par le diff comptent, pas la dette préexistante
8. Si tout est vert : `git push` de la branche de version

À vérifier après coup :

- Le commit de fusion contient bien la liste des commits intégrés (pas juste « Merge branch X into Y »).
- Aucun conflit n'a été résolu à l'aveugle sur un point métier — en cas de doute, la procédure doit s'arrêter et demander plutôt que trancher seule.
- Build/tests/lint sont bien passés **après** la fusion, pas seulement avant (l'intégration de plusieurs travaux peut casser des choses invisibles avant).

Si ça échoue : la procédure corrige sur place si c'est trivial, sinon elle s'arrête et explique — jamais de `push --force`, jamais de fusion poussée avec des tests rouges.

Étape 3 — Beta (``/release-beta``)

Déclencheur : la branche de version est jugée stable, prête pour un tour de validation interne.

Qui décide : chef de projet ou développeur senior. **Qui exécute :** développeur ou chef de projet.

Où : Claude Code, dans ``logeas-web``, **sur la branche de version** (pas sur une branche perso).

Ce que la commande fait, dans l'ordre :

1. `git status` — vérifie que la branche est propre et à jour avec origin
2. `npm run build:beta`, qui enchaîne :
 - `node scripts/version.js beta` — calcule le numéro X.Y.Z.W (voir règle de numérotation dans la procédure de référence)
 - régénération de la licence DevExtreme
 - `ng build --configuration production, beta` — sortie vers LogeasWeb-Test
3. `node scripts/release-report.js beta`, qui :
 - calcule le diff Git depuis le **dernier tag _release** (jamais depuis le dernier beta — le rapport cumule tout le cycle)
 - relance `npx jest --coverage --json` pour un résumé de tests à jour (pass/fail, couverture)
 - lit la version des 4 composants de la Galaxie LoGeAs directement sur les exécutables compilés (D:\Developpements\VersionPublic\ExecutablesWindows\)
 - écrit `src/assets/releases/X.Y.Z.W.html` + met à jour `manifest.json`
4. Commit dédié « Preparation version X.Y.Z.W » (``package.json`` + ``version.ts`` + le rapport généré)
5. `git tag -a X.Y.Z.W_beta`
6. `git push + git push origin <tag>`

À vérifier après coup :

- Le numéro de version généré est bien celui attendu (cohérent avec le nom de la branche).
- Le rapport de release ne signale pas d'échec de test bloquant — s'il y en a, décider **avec l'équipe** si on tague quand même.
- Le build est bien présent dans D:\Developpements\VersionPublic\LogeasWeb-Test\browser\.
- Le rapport est consultable dans l'appli via l'écran « Rapports de release » (Administration, ou directement /RapportsRelease, accessible sans connexion).

Ce qui reste manuel : copier LogeasWeb-Test\browser\ vers le serveur de test — accessible uniquement en bureau à distance, aucune procédure ne le fait à votre place.

Étape 4 — Release client ('/release-client')

Déclencheur : la beta a été validée (ou directement si aucune beta n'est jugée nécessaire pour ce cycle).

Qui décide : chef de projet — c'est la décision de livrer au client. **Qui exécute :** développeur ou chef de projet.

Où : Claude Code, dans ``logeas-web`` (étapes 1 à 6), puis dans ``logeas-cartographie`` pour la relecture/le commit du rapport qualité (étape 7).

Ce que la commande fait, dans l'ordre :

1. `git status` — branche propre et à jour
2. Si cette release fait suite à une beta : vérifie que rien d'autre que des correctifs ciblés n'a été ajouté depuis le tag beta (`git log <tag_beta>..HEAD`) — sinon ça mérite peut-être un nouveau tour de beta.
3. `npm run build`, qui enchaîne :
 - `node scripts/version.js release` — repasse en 3 chiffres, **reprend le même X.Y.Z que les betas du cycle** (jamais de numéro « orphelin »)
 - `ng build --configuration production` — sortie vers LogeasWeb
4. `node scripts/release-report.js release` — même mécanique qu'en beta
5. Commit dédié « Preparation version X.Y.Z » (`package.json` + `version.ts` + le rapport)
6. `git tag -a X.Y.Z_release`
7. `git push` + push du tag
8. **/rapport-tests-qualite** — uniquement à cette étape, jamais en beta : régénère le rapport qualité NF dans `logeas-cartographie` (dépôt frère, pas `logeas-web`), à partir d'une exécution réelle des tests de `logeas-web` **et** `logeas-lib`

À vérifier après coup :

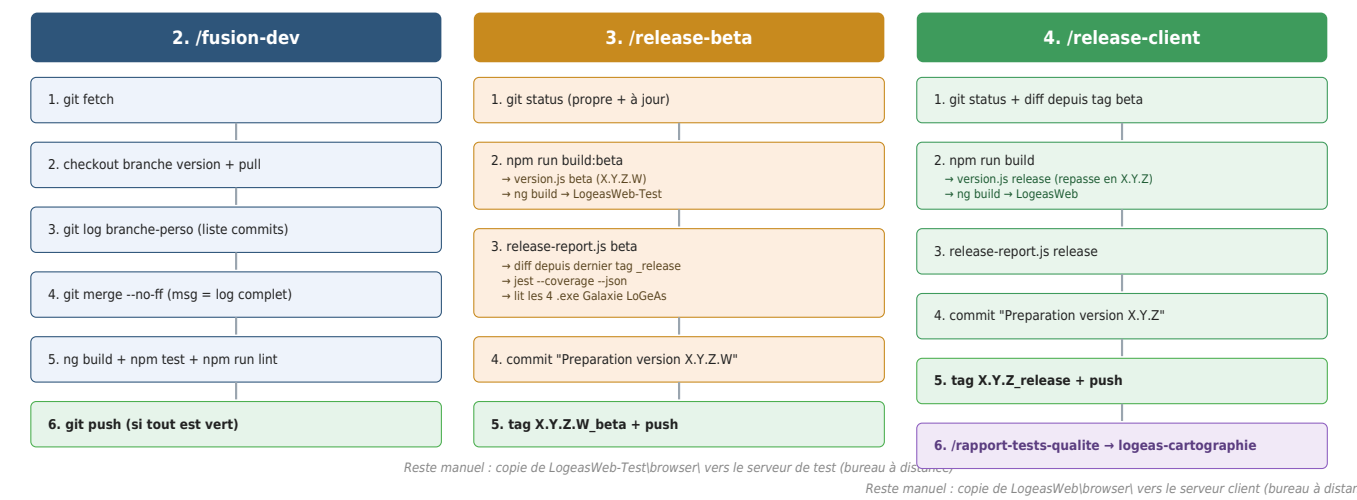
- Le build est bien présent dans `D:\Developpements\VersionPublic\LogeasWeb\browser\`.
- Le tag ne rentre pas en collision avec un `tag_release` déjà existant.
- Le rapport qualité NF (`logeas-cartographie`) a bien été régénéré, relu, et **proposé au commit** — jamais commité automatiquement, dans aucun des deux dépôts.

Ce qui reste manuel : copie vers le serveur client (bureau à distance).

Aide-mémoire : qui fait quoi, sous le capot

Ce que font les 3 commandes outillées, sous le capot

Étape 1 (travail perso) est manuelle, sans commande — non représentée ici



Le rapport qualité NF (étape 6 de release-client) tourne sur **logeas-web ET logeas-lib** — c'est pour ça qu'il n'est déclenché qu'en release, jamais à chaque beta : plusieurs minutes de tests en plus, pour un document qui n'a pas vocation à refléter chaque itération intermédiaire.

Aucune de ces commandes ne push jamais en force, ni ne commite un build cassé — en cas d'échec non trivial, elles s'arrêtent et expliquent.

Équivalences SourceTree des commandes Git

Toutes les commandes Git listées ci-dessus sont exécutées automatiquement par les commandes Claude Code — vous n'avez normalement rien à taper à la main. Ce tableau sert pour les cas où on doit reproduire une étape manuellement (dépannage, vérification, ou si on préfère l'interface graphique) :

Commande Git	Équivalent dans SourceTree
git fetch	Bouton Fetch (barre d'outils du haut)
git checkout <branche>	Onglet Branches (panneau de gauche) → double-clic sur la branche
git pull	Bouton Pull
git status	Onglet File status — visible en permanence, pas besoin de le lancer
git add <fichiers>	Onglet File status → cocher les fichiers à inclure dans le prochain commit
git commit -m "..."	Bouton Commit → saisir le message dans la zone de texte
git merge --no-ff <branche>	Clic droit sur la branche à fusionner (dans le graphe ou l'onglet Branches) → Merge [branche] into [branche courante] → cocher « Create a commit even if merge resolved via fast-forward »

Commande Git	Équivalent dans SourceTree
<code>git tag -a X.Y.Z_release -m "..."</code>	Clic droit sur le commit dans le graphe → Tag... → cocher Annotated tag , saisir le nom et le message
<code>git push</code>	Bouton Push → dans la boîte de dialogue, cocher aussi les tags à pousser (case « Push all tags » ou sélection du tag spécifique) — sinon le tag reste local
<code>git log <a>..</code>	Sélectionner la branche dans le graphe et lire l'historique visuellement ; le format exact utilisé par /fusion-dev (un séparateur technique par commit) est interne au script, pas à reproduire à la main

Point d'attention SourceTree spécifique aux tags : SourceTree ne pousse pas les tags automatiquement avec un push classique — il faut explicitement cocher le tag dans la fenêtre de push, ou faire un push dédié sur le tag après coup. C'est une source d'erreur fréquente (tag créé localement mais jamais visible sur ``origin``).

Erreurs fréquentes à connaître

- **Confondre « beta » et « alpha » :** il n'y a que deux étapes outillées de génération, beta et release. Pas de troisième étape intermédiaire.
- **Lancer `npm run build` à la main pour un simple test de compilation :** ça déclenche le bump de version (``prebuild``). Pour juste vérifier que ça compile sans effet de bord, utiliser `ng build` directement (c'est ce que fait /fusion-dev).
- **Oublier que le rapport qualité NF ne se met à jour qu'en release,** pas en beta — si besoin d'un état des lieux ponctuel entre deux releases, relancer /rapport-tests-qualite manuellement.
- **Croire que le déploiement est automatique :** à aucune étape le contenu n'est copié sur un vrai serveur — c'est toujours une action manuelle, via bureau à distance.
- **Pousser un commit sans pousser le tag associé (SourceTree) :** voir l'encadré ci-dessus — un tag local qui n'existe pas sur ``origin`` ne sert à rien pour l'équipe.

Document généré le 11 août 2026, à mettre à jour si le contenu des commandes /fusion-dev, /release-beta, /release-client ou /rapport-tests-qualite évolue.

From:
<https://wiki-logeas.fr/certif/> - dokuwiki-certif

Permanent link:
<https://wiki-logeas.fr/certif/doku.php?id=certif:procedure:develop:gestionversionpratique&rev=1786463948>

Last update: 2026/08/11 17:59

