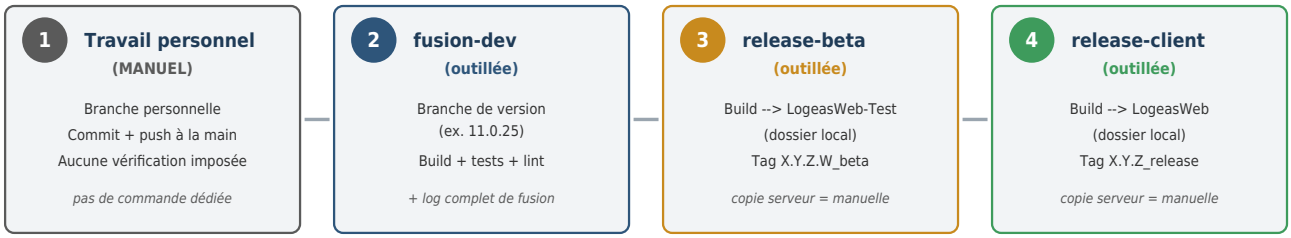
Guide pratique — Cycle de release, étape par étape

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

Vue d'ensemble



Aucune étape ne crée de nouvelle branche Git pour la beta ou la release — un simple tag marque le code correspondant.

Étape	Qui / quoi	Commande
1. Travail personnel	Développeur, à la main	(aucune — voir détail plus bas)
2. Intégration	Développeur	`/fusion-dev`
3. Beta	Développeur ou chef de projet	`/release-beta`
4. Release client	Chef de projet (validation finale)	`/release-client`

Étape 1 — Travail personnel (manuel)

Déclencheur : en continu, dès qu'on commence à développer quelque chose.

Ce qu'on fait, à la main :

1. `git checkout -b <nom-de-branche>` depuis la branche de version en cours (ex. `11.0.25`) — pas de convention de nom imposée.
2. On développe, on teste localement si besoin.
3. `git add <fichiers>` — **jamais** `git add -A` à l'aveugle, on relit ce qu'on ajoute.
4. `git commit -m "…"` — message en français, court, orienté fonctionnel.
5. `git push -u origin <branche>` — sauvegarde régulière, au fil de l'eau.

Volontairement aucune vérification ici (pas de build, pas de tests) : c'est une sauvegarde de 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.

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.

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).

Ce que la commande fait, dans l'ordre :

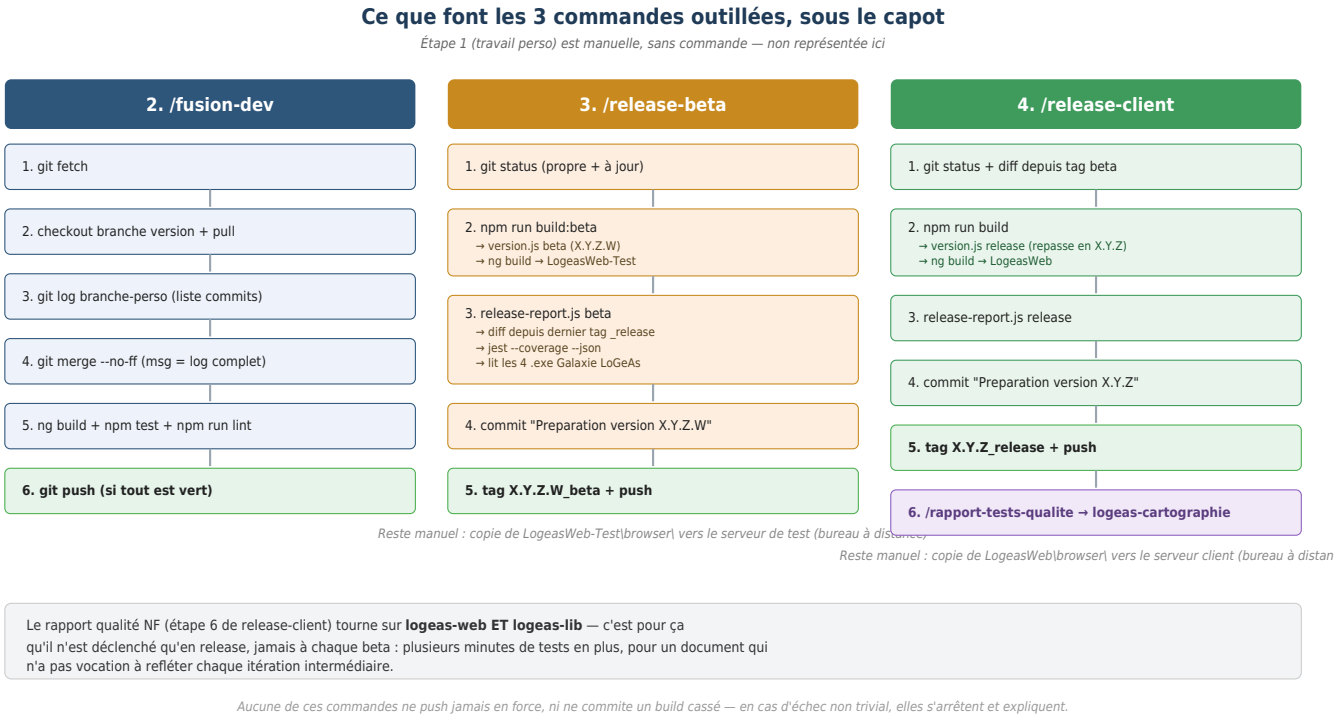
1. `git status` — branche propre et à jour
2. Si cette release fait suite à une beta : vérifie que rien d'autre que des correctifs ciblés n'a été ajouté depuis le tag beta (`git log <tag_beta>..HEAD`) — sinon ça mérite peut-être un nouveau tour de beta.
3. `npm run build`, qui enchaîne :
   - `node scripts/version.js 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



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.

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=1786463295

Last update: **2026/08/11 17:48**