SVN pour les développeurs habitués à Git

Dans l'industrie automobile, aérospatiale et dans certains environnements de test embarqués, SVN (Subversion) reste largement utilisé — souvent en parallèle de DOORS pour la gestion d'exigences et de Jenkins pour l'intégration continue. Si vous venez du monde web où Git a presque totalement supplanté SVN, voici ce qu'il faut savoir pour être opérationnel rapidement.

Pourquoi SVN est encore vivant en 2026

Contrairement à une idée reçue, SVN n'est pas un outil mort. Il reste pertinent dans les contextes où :

- la conformité réglementaire (DO-178C, ISO 26262, ASPICE) exige une traçabilité centralisée simple à auditer
- les fichiers binaires volumineux (firmwares, modèles CAO) sont gérés au quotidien — SVN les verrouille mieux que Git
- les équipes ont besoin d'un workflow centralisé, sans la complexité du DAG de Git

Différence fondamentale : centralisé vs décentralisé

Git est décentralisé : chaque commit reste local jusqu'à un `git push`. Vous avez l'historique complet du dépôt sur votre machine.

SVN est centralisé : il n'existe qu'un seul dépôt central. Un `svn commit` envoie directement vos changements au serveur — il n'y a pas d'étape intermédiaire locale.

```
# Git — deux étapes
git commit -m "message"
git push

# SVN — une seule étape
svn commit -m "message"
```

Commandes essentielles

Action	Git	SVN
Récupérer le dépôt	`git clone`	`svn checkout`
Suivre un fichier	`git add`	`svn add`
Envoyer les changements	`git commit` + `git push`	`svn commit`
Voir les changements	`git status`	`svn status`
Voir le détail	`git diff`	`svn diff`
Récupérer les changements distants	`git pull`	`svn update`

Historique	`git log`	`svn log`
Annuler localement	`git checkout --`	`svn revert`

(Si vous lisez ceci en dehors d'un environnement supportant les tableaux Markdown, gardez cette liste sous forme de liste à puces — un tableau peut mal s'afficher selon l'éditeur.)

Les révisions sont globales, pas par commit

C'est la première vraie surprise. En Git, chaque commit a un hash unique indépendant des autres fichiers. En SVN, chaque commit incrémente un **numéro de révision global** sur tout le dépôt — peu importe si vous modifiez un fichier ou cinquante.

```
svn commit -m "Ajout de deux fichiers"
# Committed revision 1.

svn commit -m "Modification d'un fichier"
# Committed revision 2.
```

Vous ne réutiliserez jamais le numéro 1, même si vous ajoutez ensuite cent autres fichiers.

Gestion des conflits

Quand deux personnes modifient le même fichier sans synchroniser, `svn update` détecte le conflit et vous propose un menu interactif :

```
Select: (p) Postpone, (df) Show diff, (e) Edit file, (m) Merge,
        (s) Show all options:
```

En choisissant `p` (Postpone), SVN insère des marqueurs dans le fichier :

```
<<<<<<< .mine
Votre version locale non commitée
||||||| .r2
La version commune avant divergence
=======
La version déjà commitée par l'autre personne
>>>>>>> .r3
```

Notez la section `|||||||` — c'est une vraie différence avec Git, qui n'affiche par défaut que les deux versions en conflit sans la base commune intermédiaire.

Une fois les marqueurs supprimés manuellement et le contenu final choisi :

```
svn resolve --accept working nom_du_fichier.md
svn commit -m "Résolution du conflit"
```

Piège à éviter : `svn resolve --accept working` ne nettoie pas le contenu automatiquement, il marque juste le conflit comme réglé avec ce qu'il y a déjà dans le fichier à ce moment-là. Supprimez les marqueurs avant de lancer `resolve`, pas après.

Branches : des copies physiques, pas des pointeurs

En Git, une branche est un pointeur léger vers un commit. En SVN, une branche est une **copie complète** du dépôt, créée par convention dans un dossier `branches/` :

```
svn copy file:///chemin/vers/depot/trunk \
         file:///chemin/vers/depot/branches/ma-feature \
         -m "Création de la branche ma-feature"
```

La structure conventionnelle d'un dépôt SVN est :

```
mon-depot/
├── trunk/        # ligne principale (équivalent de main/master)
├── branches/     # branches de travail
└── tags/         # versions figées
```

On checkoute ensuite cette branche comme un dépôt normal :

```
svn checkout file:///chemin/vers/depot/branches/ma-feature copie-locale
```

Merge : deux temps, pas un seul

Contrairement à Git où `git merge` agit directement entre branches, en SVN le merge se fait dans une copie de travail, puis se commit comme une révision normale :

```
cd trunk-local
svn update
svn merge file:///chemin/vers/depot/branches/ma-feature
svn commit -m "Merge de ma-feature vers trunk"
```

Cherry-pick : récupérer une seule révision

Très utile en contexte industriel pour appliquer un correctif urgent sans embarquer une fonctionnalité pas encore prête sur la même branche :

```
svn merge -c 9 file:///chemin/vers/depot/branches/ma-feature
```

Le `-c 9` ne récupère que la révision 9, en ignorant toutes les autres révisions de la branche.

svn:ignore : une propriété, pas un fichier

SVN n'a pas de fichier `.gitignore`. À la place, il pose une **propriété** sur un dossier :

```
svn propset svn:ignore "*.log" .
svn commit -m "Ajout de svn:ignore"
```

Cette propriété doit elle-même être commitée — c'est une métadonnée versionnée comme le reste.

Tags : une convention, pas une contrainte technique

Un tag SVN n'est rien d'autre qu'un `svn copy` vers le dossier `tags/` :

```
svn copy file:///chemin/vers/depot/trunk \
         file:///chemin/vers/depot/tags/v1.0 \
         -m "Tag version 1.0"
```

Attention : contrairement à un tag Git qui est réellement immuable, un tag SVN peut techniquement être modifié — c'est une pure convention d'équipe à respecter, pas une protection du système.

svn:externals : l'équivalent des submodules Git

Pour inclure un dépôt externe (une bibliothèque partagée entre plusieurs projets, par exemple), on pose la propriété `svn:externals` :

```
echo "shared-lib -r1 file:///chemin/vers/depot-partage" > externals.txt
svn propset svn:externals -F externals.txt .
svn commit -m "Ajout de l'external shared-lib"
svn update
```

Sans le `-r1`, l'external suit toujours la dernière révision du dépôt externe. Avec une révision fixée, le comportement se rapproche de celui d'un submodule Git pointant vers un commit précis.

Piège de syntaxe courant : l'ordre des arguments compte. La syntaxe correcte est `nom_dossier -rX url`, pas `-rX nom_dossier url`. Et la valeur de la propriété ne doit contenir aucun guillemet — passez par un fichier texte (`-F`) plutôt que par un argument shell direct pour éviter les erreurs de parsing.

Récapitulatif

Concept	Git	SVN
Architecture	Décentralisé	Centralisé
Identifiant de commit	Hash unique	Révision globale incrémentale
Branches	Pointeurs légers	Copies physiques (`svn copy`)
Fichiers ignorés	`.gitignore` (fichier)	`svn:ignore` (propriété)
Sous-modules	`.gitmodules`	`svn:externals` (propriété)
Tags	Réellement immuables	Convention, pas de contrainte technique

SVN demande un changement de mentalité plus qu'un changement de compétence : on perd la liberté du travail local complet de Git, mais on gagne en simplicité d'audit — un vrai atout dans les environnements réglementés.