

Sessions garanties disponibles

Formation Git Avancé : Internals, réécriture d'historique et dépôts complexes

Réf. SF34631

Allez sous le capot de Git pour maîtriser les scénarios que le quotidien ne suffit pas à enseigner : réécriture d'historique, conflits récurrents, gros dépôts et dépendances inter-projets.

2 j Durée	1 600 € HT Inter / participant	3 700 € HT Intra (≤12 pers.)	1 Sessions à venir
--------------	-----------------------------------	---------------------------------	-----------------------

Objectifs pédagogiques

- ✓ Expliquer le modèle objet et les internals de Git pour diagnostiquer le comportement de l'outil sans recourir à la magie
- ✓ Manipuler et réécrire l'historique en profondeur pour récupérer une situation ou nettoyer un dépôt
- ✓ Résoudre les conflits complexes en exploitant les stratégies de merge avancées et `git rerere` pour automatiser les résolutions répétitives
- ✓ Configurer le comportement de Git au niveau fichier avec `.gitattributes` et annoter l'historique avec `git notes`
- ✓ Mettre en œuvre un workflow de patches pour collaborer hors plateforme
- ✓ Orchestrer plusieurs arbres ou dépôts avec `git worktree`, sous-modules et subtrees, et choisir l'approche adaptée à chaque contexte
- ✓ Gérer des dépôts volumineux ou à historique long avec Git LFS, `sparse-checkout` et `partial clone`

Public visé & prérequis

Public visé Architectes, Développeurs

Prérequis- Maîtrise des commandes Git du quotidien : `add`, `commit`, `push`, `pull`, `fetch`, `stash`, `cherry-pick` - Pratique régulière du travail en branches : création, switch, merge, rebase interactif (squash/fixup), résolution de conflits - Expérience du travail collaboratif distant : pull/merge request, revue de code, workflow d'équipe - Avoir suivi la formation Sparks Git de niveau 1 ou disposer d'une expérience Git équivalente en production

Programme détaillé

Modèle objet et internals de Git

- Anatomie du dossier `.git` : objects, refs, HEAD, index, packfiles — où vit réellement le code
- Les quatre objets de Git : blob, tree, commit, tag — au-delà du survol, lecture brute avec les commandes de plomberie
- Commandes plumbing vs porcelain : `cat-file`, `hash-object`, `ls-tree`, `rev-parse` — diagnostiquer sans magie
- Refs et refsspecs : branches locales, branches de suivi, tags, refs distantes — lecture et manipulation fine
- Packfiles et garbage collection : `gc`, `prune`, `fsck` — entretenir un dépôt et comprendre la compression
- Reachability et orphan commits : ce que Git « voit » et ce qu'il efface — fondement de la récupération d'historique

Activités pratiques :

- Reconstitution manuelle d'un commit à partir de blobs et trees avec les commandes plumbing
- Manipulation d'un refspect personnalisé pour ne récupérer qu'un sous-ensemble de branches distantes
- Diagnostic d'un dépôt corrompu et exécution d'un `gc` agressif

Réécriture et maintenance avancée de l'historique

- reflog` en profondeur : journal local de toutes les manipulations de refs — le filet de sécurité du quotidien
- Récupération d'une branche supprimée ou d'un commit perdu : reconstruction via le reflog et les commits orphelins

- ``rebase –autosquash`` et ``commit –fixup`` : workflow propre pour rectifier au fil de l'eau sans rebase manuel
- ``rebase –onto`` : déplacer une série de commits d'une base à une autre — cas du backport ou du rebranchage
- ``git filter-repo`` : outil moderne de réécriture d'historique en profondeur (purge de fichier, scission de dépôt, refonte d'identités) — positionné comme remplaçant de ``filter-branch`` (mention historique uniquement)
- ``git bisect`` : recherche dichotomique d'une régression dans un historique long — automatisable avec ``bisect run``
- `Range-diff` et ``cherry-pick -x`` : comparer deux séries de commits, tracer l'origine d'un cherry-pick

Activités pratiques :

- Récupération d'une branche supprimée par erreur via le reflog
- Purge définitive d'un fichier de configuration contenant un secret commit accidentellement, sur tout l'historique
- Chasse automatisée à une régression avec ``git bisect run`` couplé à un script de test
- Scission d'un sous-dossier d'un dépôt en un dépôt indépendant avec ``git filter-repo``

Conflits complexes et stratégies de fusion

- Anatomie d'une fusion : fast-forward, three-way merge, base commune — mécanique sous les marqueurs de conflit
- Stratégies de merge : ``recursive``, ``ort`` (par défaut depuis Git 2.34), ``ours``, ``theirs`` — quand utiliser laquelle
- Options de stratégie : ``-X ignore-space-change``, ``-X patience``, ``-X diff-algorithm`` — désamorcer les conflits artificiels
- ``git rerere`` (reuse recorded resolution) : enregistrer une résolution de conflit pour la rejouer automatiquement lors d'un rebase ou d'une re-fusion
- Annulation propre d'un merge déjà poussé : ``revert -m``, ré-application ultérieure et pièges associés
- Diagnostic d'un conflit : ``git log –merge``, ``git diff –base``, outils de mergetool — comprendre l'origine du désaccord

Activités pratiques :

- Résolution d'un conflit récurrent sur une branche longue avec activation de ``rerere``
- Comparaison des résultats d'un même merge avec stratégies et options différentes
- Annulation d'un merge poussé sur main, puis ré-application contrôlée du contenu

Configuration fine au niveau fichier : attributes et notes

- `.gitattributes`` : principe et portée — règles attachées aux chemins de fichiers
- Gestion des fins de ligne et de l'encodage : ``text``, ``eol``, ``working-tree-encoding`` — éviter les diffs parasites en équipe cross-OS
- Diff et merge personnalisés : ``diff.driver``, ``merge.driver`` — rendre lisibles les diffs de fichiers binaires (Word, images) ou de formats structurés
- Filtres ``clean`` et ``smudge`` : transformer le contenu à l'entrée/sortie du dépôt — usage pour secrets, formatage, normalisation
- ``export-ignore`` et ``linguist-*`` : contrôler les exports d'archive et la détection de langage par GitHub/GitLab
- ``git notes`` : annoter un commit sans réécrire son hash — usage pour CI, code review, classement par catégorie
- Partage des notes : refsspecs dédiées, synchronisation avec un remote

Activités pratiques :

- Mise en place d'un driver de diff lisible pour un fichier Word ou un format JSON ordonné
- Configuration d'un filtre clean/smudge pour normaliser un fichier de configuration avant commit
- Ajout, classement par catégorie et partage de notes sur l'historique d'un dépôt existant

Patches et collaboration hors plateforme

- Pourquoi les patches : scénarios de contribution sans GitHub/GitLab (kernel-style, échange inter-organisations, dépôts cloisonnés)
- ``git format-patch`` : génération d'une série de patches à partir d'un ou plusieurs commits — anatomie d'un fichier ``.patch``
- ``git apply`` vs ``git am`` : appliquer un patch sans historique vs intégrer une série en préservant l'auteur et le message
- Workflow complet : préparer une série, l'envoyer, la recevoir, l'appliquer, gérer les conflits
- Bundles Git : ``git bundle`` pour échanger un dépôt entier ou une portion sans réseau partagé
- Bonnes pratiques : tester un patch avec ``–check``, signer un patch, gérer les rejets

Activités pratiques :

- Génération d'une série de patches d'une feature et application sur un dépôt cible avec ``git am``
- Création d'un bundle Git pour échanger un sous-ensemble du dépôt
- Gestion d'un conflit lors de l'application d'un patch et résolution avec ``git am –continue``

Multi-arborescence et dépôts composites

- `git worktree`` : plusieurs branches checkout-ées en parallèle sans clone — gain quand on jongle entre hotfix et feature
- Cycle de vie d'un worktree : ``add``, ``list``, ``lock``, ``move``, ``remove`` — entretien et nettoyage
- Sous-modules : intégrer un dépôt dans un dépôt — ajout, clone récursif, mise à jour, suppression propre

- Pièges des sous-modules : commits détachés, mises à jour à coordonner, branches non suivies — pratiques pour les éviter
- Subtrees : alternative aux sous-modules par fusion d'histoires — ``git subtree add``, ``pull``, ``push``, ``split``
- Arbitrer sous-module vs subtree vs monorepo : critères de choix (couplage, propriété, fréquence de mise à jour, taille)

Activités pratiques :

- Mise en place d'un worktree pour traiter un hotfix sans abandonner la branche en cours
- Ajout d'un sous-module à un dépôt existant, clonage récursif et mise à jour vers une nouvelle version
- Comparaison sur un même cas concret de l'intégration via sous-module et via subtree
- Choix raisonné sous-module / subtree / monorepo sur un scénario projet donné

Gros dépôts et performance

- Anatomie d'un dépôt qui devient lourd : binaires, longue histoire, monorepo — symptômes (clone lent, opérations qui rament)
- Git LFS (Large File Storage) : principe pointer-store, installation et configuration côté client, ``git lfs track``
- Workflow LFS au quotidien : commits, fetch, push, migration d'historique existant vers LFS
- Clones partiels et superficiels : ``--depth``, ``--shallow-since``, ``--single-branch`` — quand et comment unshallow
- ``--filter=blob:none`` et ``partial clone`` : récupérer la structure sans tous les blobs, fetch à la demande
- ``sparse-checkout`` : ne matérialiser qu'une partie de l'arborescence — modes cone et non-cone
- Combinaison ``partial clone`` + ``sparse-checkout`` : pattern recommandé pour gros monorepos

Activités pratiques :

- Mise en place de Git LFS sur un dépôt contenant des assets binaires et migration de l'historique
- Réalisation d'un partial clone d'un gros dépôt et observation des fetches à la demande
- Configuration d'un ``sparse-checkout`` pour ne récupérer qu'un sous-projet d'un monorepo

Prochaines sessions

DATE	LIEU	STATUT
12/11/2026	à distance	Sur confirmation

Pourquoi choisir Sparks Formation ?

20 ans d'expertise IT, des formateurs experts et des sessions garanties contre l'annulation.

20 ans
d'expertise

1 068+
formations

11 178
apprenants formés/an

+67
Net Promoter Score

Avis des participants

Note moyenne des participants de la filière CI/CD & Usine Logicielle : 4,9/5 — 24 avis

Dorian B. — Formateur à l'écoute, curieux et qui livre des retours d'expériences intéressants.

Patrice B. — Formation très intéressante et intense. Formateur très compétent et à l'écoute.

Ils nous font confiance

Cette année notre filière CI/CD & Usine Logicielle a accompagné plus de 102 entreprises dont :

INETUM SII ILE DE FRANCE LA FRANÇAISE DES JEUX AMADEUS SAS HARDIS GROUP SILAE AMADEUS DATA PROCESSING GMBH EDF EPSILON FRANCE FDJ-UNITED +92 autres

Financement & certifications

Cette formation est éligible aux financements OPCO et FNE-Formation. Nos équipes vous accompagnent dans le montage de votre dossier.



Réservez votre place dès maintenant

20 ans d'expertise · 1 000+ formations · NPS +67 · réponse sous 24h ouvrées

0 805 950 800

demande@sparks-formation.com