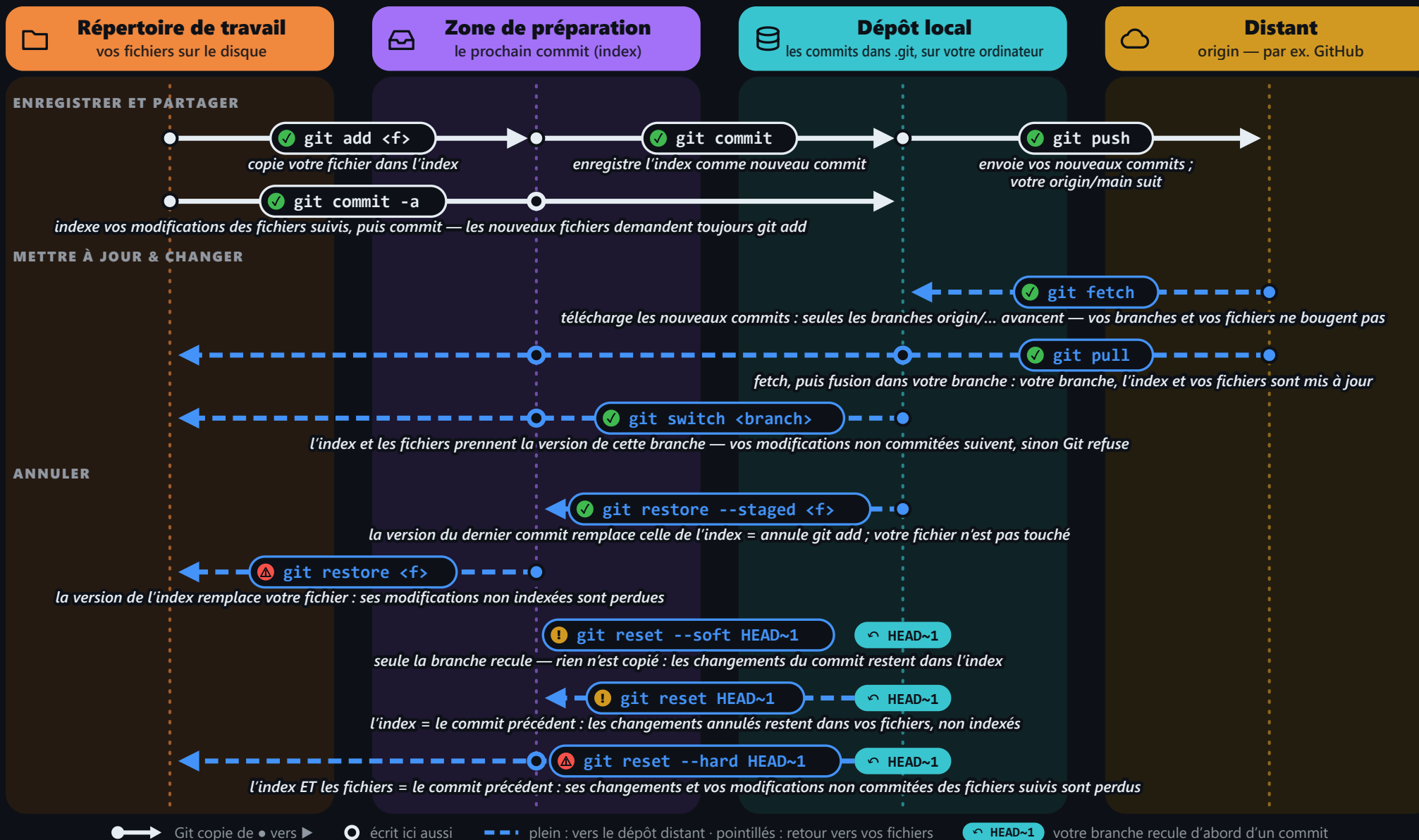


01 Git en un coup d'œil

Les quatre endroits où Git garde votre travail — et ce que chaque commande copie exactement entre eux

✓ Sans risque ⚠ Réécrit l'historique ⚠ Peut faire perdre du travail



o **<f>** un chemin de fichier : tapez **app.js**, sans les < >

o **HEAD~1** le commit avant celui où vous êtes

o **git pull** si vous et le dépôt distant avez tous deux de nouveaux commits, Git s'arrête jusqu'à ce que vous choisissiez : **git config pull.rebase false** (fusion) ou **true** (rebase)

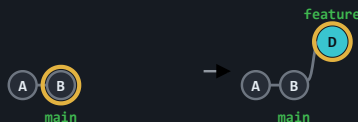
Avant → après. Les cercles sont des commits, l'anneau jaune indique où vous êtes (HEAD). ● nouveau commit ○ plus aucune branche ne pointe dessus

✓ git commit



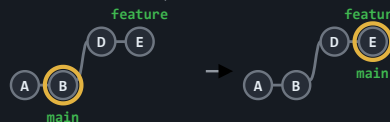
Un nouveau commit à partir de la préparation ; votre branche avance dessus.

✓ git switch -c feature + commit



Une nouvelle branche part de là où vous êtes ; vos prochains commits la font avancer, main ne bouge pas.

✓ git merge feature (fast-forward)



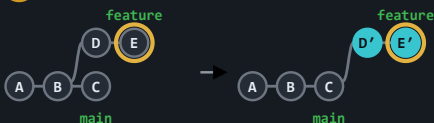
Sur main, quand main n'a pas de nouveaux commits : main avance simplement jusqu'à E — pas de commit de fusion.

✓ git merge feature



Sur main : intègre feature avec un commit de fusion qui a deux parents.

! git rebase main



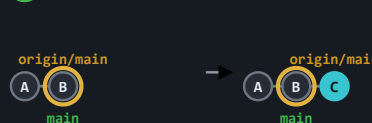
Sur feature : rejoue D et E par-dessus main, en copies avec de nouveaux hash (D' E') ; les originaux quittent la branche.

✓ git cherry-pick D



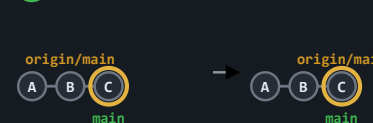
Copie un commit sur votre branche (D', nouveau hash) ; l'original reste en place.

✓ git fetch



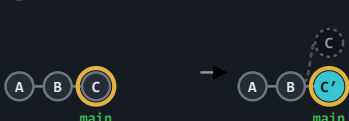
Télécharge les nouveaux commits : seules les étiquettes origin/... avancent. Votre branche et vos fichiers ne bougent pas.

✓ git push



Envoie C : main sur le dépôt distant — et votre origin/main — passent à C. Vos fichiers ne changent pas.

! git commit --amend



Remplace le dernier commit par un nouveau (C', nouveau hash) ; C quitte la branche. À faire avant de pousser.

✓ git revert C



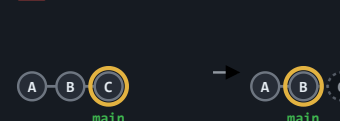
Ajoute un commit R qui annule C. L'historique est conservé : sans risque sur une branche partagée.

! git reset --soft HEAD~1



Ramène votre branche à B ; les changements de C restent dans l'index — refaites un commit pour le recréer.

⚠ git reset --hard HEAD~1



Fait reculer votre branche ; l'index et les fichiers deviennent B : les changements de C ont disparu (git reflog retrouve encore C).



Sans risque



Réécrit l'historique



Peut faire perdre du travail

DÉMARRER

- git config --global user.name "Your Name"**
Nom inscrit dans vos commits (une fois par ordinateur)
- git config --global user.email "you@example.com"**
E-mail inscrit dans vos commits
- git init**
Transformer le dossier courant en dépôt
- git clone <url>**
Télécharger un dépôt avec tout son historique

OBSERVER

- git status**
Ce qui a changé, ce qui est préparé, sur quelle branche vous êtes
- git diff**
Modifications pas encore préparées (répertoire de travail vs zone de préparation)
- git diff --staged**
Ce que contiendra le prochain commit
- git log --oneline --graph --all**
L'historique de chaque branche, une ligne par commit
- git show <commit>**
Un commit : message et modifications

ENREGISTRER VOTRE TRAVAIL

- git add <file>**
Préparer un fichier pour le prochain commit
- git add -p**
Préparer seulement certaines parties (hunks)
- git commit -m "message"**
Enregistrer ce qui est préparé sous forme de commit
- git commit --amend**
! Refaire le dernier commit (message ou contenu)
- git restore <file>**
⚠ Jeter les modifications non préparées d'un fichier
- git restore --staged <file>**
Retirer de la préparation (votre modification reste sur le disque)

PARTAGER

- git fetch**
Télécharger les nouveaux commits ; vos branches ne bougent pas
- git pull**
Fetch, puis fusion dans la branche courante
- git pull --rebase**
! Fetch, puis rejoue vos commits par-dessus (nouveaux hash)
- git push**
Envoyer vos commits vers l'upstream de la branche
- git push -u origin <branch>**
Premier push d'une nouvelle branche (définit l'upstream)
- git remote -v**
Avec quels dépôts distants ce dépôt communique

- o **HEAD** le commit où vous êtes (en général le dernier de votre branche actuelle)
- o **origin** le nom par défaut du dépôt distant que vous avez cloné
- o **<file>** un chemin de fichier : tapez **app.js**, sans les **< >**

CRÉER ET BASCULER

- git branch**
Lister les branches locales (* = actuelle)
- git branch -a**
Lister aussi les branches de suivi distantes
- git switch <branch>**
Aller sur une branche (les fichiers suivent)
- git switch -c <branch>**
Créer une branche ici et basculer dessus
- git switch -**
Revenir à la branche précédente
- git branch -m <new-name>**
Renommer la branche courante

COMBINER

- git merge <branch>**
Intégrer une branche dans la branche courante
- git merge --no-ff <branch>**
Toujours créer un commit de fusion
- git rebase <branch>**
! Rejouer vos commits par-dessus <branch> (nouveaux hashes)
- git rebase -i HEAD~3**
! Modifier les 3 derniers commits : réordonner, fusionner, reformuler, supprimer
- git cherry-pick <commit>**
Copier un commit sur la branche courante

CONFLITS

- git status**
Liste les fichiers en conflit (par ex. « both modified »)
- Modifier le fichier**
Gardez ce que vous voulez, supprimez les lignes
<<<<<< ===== >>>>>>
- git add <file>**
Marquer le fichier comme résolu
- git merge --continue**
Terminer (ou : git rebase --continue / cherry-pick -continue)
- git merge --abort**
Abandonner et revenir à l'état d'avant (rebase --abort, ...)

FAIRE LE MÉNAGE

- git branch -d <branch>**
Supprimer une branche fusionnée (refuse si elle n'est pas fusionnée)
- git branch -D <branch>**
⚠ Supprimer même si elle n'est pas fusionnée
- git push origin --delete <branch>**
⚠ Supprime la branche sur le dépôt distant (ses commits non fusionnés partent avec)
- git fetch --prune**
Supprime aussi les copies origin/... des branches supprimées sur le dépôt distant

TAGS

- git tag**
Lister les tags
- git tag -a v1.0 -m "Release 1.0"**
Tag annoté sur le commit courant
- git push origin v1.0**
Les tags ne sont pas poussés automatiquement
- git push --follow-tags**
Pousser les commits et leurs tags annotés

LIRE L'HISTORIQUE

- git log --oneline**
Historique compact de la branche courante
- git log -p <file>**
Chaque modification apportée à un fichier
- git log main..feature**
Les commits de feature que main n'a pas
- git blame <file>**
Qui a modifié chaque ligne en dernier

JE VEUX...	EXÉCUTER	BON À SAVOIR
❌ Jeter mes modifications d'un fichier	<code>git restore <file></code>	⚠ Les modifications non préparées sont perdues pour de bon
✅ Retirer un fichier de la préparation (garder ma modification)	<code>git restore --staged <file></code>	✓ Seule la zone de préparation change
⚠ Corriger le message du dernier commit ou ajouter un fichier oublié	<code>git add <file> && git commit --amend</code>	! Nouveau hash — à faire avant de pousser
⚠ Annuler le dernier commit, garder les modifications préparées	<code>git reset --soft HEAD~1</code>	! Votre travail reste dans l'index, prêt à être recommité — pas après un push
⚠ Annuler le dernier commit, garder les modifications dans les fichiers	<code>git reset HEAD~1</code>	! --mixed par défaut : hors de l'index mais sur le disque — pas après un push
✅ Annuler un commit déjà poussé	<code>git revert <commit></code>	✓ Ajoute un nouveau commit qui l'annule — sans risque sur les branches partagées
✅ Mettre mes modifications de côté pour l'instant	<code>git stash push -m "wip"</code>	✓ Récupérez-les avec <code>git stash pop</code> — les nouveaux fichiers non suivis demandent -u
❌ Jeter le dernier commit et ses modifications	<code>git reset --hard HEAD~1</code>	⚠ Ses changements et vos modifications non commitées des fichiers suivis sont perdus (git reflog retrouve encore le commit)
✅ Retrouver un commit « perdu » (après reset, rebase, branche supprimée)	<code>git reflog</code>	✓ Chaque endroit où HEAD est passé — puis <code>git branch rescue <hash></code>
✅ Arrêter un merge / rebase / cherry-pick en cours	<code>git merge --abort</code>	✓ Ou <code>rebase --abort</code> , <code>cherry-pick --abort</code>
⚠ Remplacer la branche distante après avoir réécrit l'historique	<code>git push --force-with-lease</code>	! Refuse si la branche distante a changé depuis votre dernier fetch

STASH

git stash

Mettre de côté les modifications non commitées

git stash -u

Inclure les fichiers non suivis (nouveaux)

git stash list

Tous les stashes, le plus récent = `stash@{0}`

git stash pop

Réapplique le plus récent et le supprime (conservé en cas de conflit)

git stash apply stash@{1}

En réappliquer un et le garder dans la liste

RESET — TROIS FORCES

git reset --soft <commit>

! Déplace la branche ; préparation et fichiers intacts

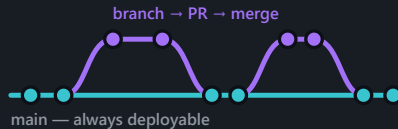
git reset <commit>

! Déplace la branche et réinitialise la préparation ; fichiers intacts

git reset --hard <commit>

⚠ Déplace la branche, la préparation ET les fichiers

GITHUB FLOW



```
git switch main && git pull
```

Partir d'un main à jour

```
git switch -c <feature>
```

Une branche de courte durée par modification

```
git commit -am "..." && git push -u origin <feature>
```

Commit, push, ouvrir une pull request

Fusionner la pull request sur GitHub

Après la revue ; puis supprimer la branche

```
git switch main && git pull --prune
```

Retour sur main ; supprime aussi les copies origin/... des branches supprimées

GIT FLOW



```
git checkout -b feature/x develop
```

Les fonctionnalités partent de develop

```
git checkout develop && git merge --no-ff feature/x
```

Terminer une fonctionnalité dans develop

```
git checkout -b release-1.1 develop
```

Geler une version

```
git checkout main && git merge --no-ff release-1.1
```

Publier sur main...

```
git tag -a 1.1 -m "Release 1.1"
```

...la taguer, puis la fusionner dans develop

TRUNK-BASED



```
git commit -am "small change"
```

De petits commits complets sur main

```
git pull --rebase
```

! Rejoue les vôtres par-dessus ceux des autres (nouveaux hash)

```
git add <file> && git rebase --continue
```

Après avoir réglé un conflit : le marquer résolu, continuer

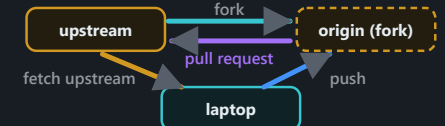
```
git push
```

Intégrer au moins une fois par jour

Livrer le travail inachevé désactivé

Derrière un feature flag, par ex. NEW_CHECKOUT = false

FORKING



```
git remote add upstream
```

```
<original-url>
```

Une seule fois : l'original comme second dépôt distant

```
git fetch upstream && git merge upstream/main
```

Synchroniser votre main avec l'original

```
git push
```

Mettre à jour votre fork (origin)

```
git switch -c <fix> ... git push -u origin <fix>
```

Travailler sur une branche de votre fork

Ouvrir une pull request vers l'original

Les mainteneurs relisent et fusionnent