

Workflows Git — aperçu et comparatif

Emmanuel BRUNO

Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

2026-10-14

Table des matières

0.1 Introduction — pourquoi des workflows ?	1
0.2 Vocabulaire clé	1
0.3 1) Workflow centralisé (style SVN) ..	1
0.4 2) Feature branch workflow (branche par fonctionnalité)	2
0.5 3) Gitflow (Vincent Driessen)	2
0.6 4) GitHub Flow (simplifié, PR-based)	2
0.7 5) Forking workflow (open-source) .	3
0.8 6) Trunk-based development (TBD) .	3
0.9 Comparatif rapide	3
0.10 Choix pragmatique	3
0.11 Intégration CI/CD	3
0.12 Bonnes pratiques transversales . . .	3
0.13 Exercices pratiques (atelier)	3
0.14 Commandes utiles (résumé)	4
0.15 Pièges fréquents	4
0.16 Ressources et lecture	4
0.17 Conclusion & prochaines étapes . . .	4
1 Fin	4

0.1 Introduction — pourquoi des workflows ?

- Un workflow définit comment une équipe utilise Git (branches, merges, PRs, releases).
- Objectifs : clarté, collaboration, qualité, intégration continue.

0.2 Vocabulaire clé

- Workflow : ensemble de pratiques et règles pour utiliser Git.
- pull request (PR) : demande de fusion d'une branche dans une autre, souvent avec revue de code.
- merge request : synonyme de PR (GitLab).
- CI : intégration continue, automatisation des tests et builds, déclenchée par des pushes/PRs.
- CD : livraison continue, déploiement automatisé en production, souvent après CI réussie.

0.3 1) Workflow centralisé (style SVN)

- Idée : une branche principale (`main/master`) et tout le monde push/pull dessus.

Diagramme :

```
Developers -> commit -> push -> origin/main
```

Commandes typiques :

```
git checkout main
git pull
git add .
git commit -m "... "
git push origin main
```

Avantages : simple, facile à comprendre pour débutants. Inconvénients : risque élevé de conflits, intégrité limitée, pas de revue obligatoire.

0.4 2) Feature branch workflow (branche par fonctionnalité)

- Chaque fonctionnalité ou bugfix a sa propre branche `feature/x`.
- Fusion via merge ou PR dans `main`.

Diagramme :

```
main ---o-----o-----o
      \         /
feature1 o-----o
```

Commandes :

```
# créer et basculer
git checkout -b feature/ma-fonctionnalite
# pousser
git push -u origin feature/ma-fonctionnalite
# après review et tests
git checkout main
git merge --no-ff feature/ma-fonctionnalite
git push origin main
```

Avantages : isole le travail, facilite revue et CI. Inconvénients : branches longues peuvent diverger; nécessité de synchroniser (rebase/merge).

0.5 3) Gitflow (Vincent Driessen)

- Branches principales : `main` (release) et `develop` (intégration)
- Branches support : `feature/`, `release/`, `hotfix/`

Diagramme :

```
main ----o-----o
        \       /
develop ---o--o--o
        /       \
feature1         release
```

Commandes (exemple simplifié) :

```
# feature
git checkout develop
git checkout -b feature/x
# push feature
git push -u origin feature/x
# finish feature
git checkout develop
git merge --no-ff feature/x
# release
git checkout -b release/1.2 develop
git checkout main
git merge --no-ff release/1.2
git tag -a v1.2
```

Avantages : clair pour projets avec versions/stabilisation. Conserve historique de releases. Inconvénients : plus complexe, lourdeur pour déploiements rapides.

0.6 4) GitHub Flow (simplifié, PR-based)

- Branche courte depuis `main`, ouvrir Pull Request, revue, merge, déploiement continu.
- Pas de `develop` ni branche de release.

Diagramme :

```
main ---o---o---o
      \   /
      PR-merge
```

Bon pour équipes produisant déploiements fréquents (CI/CD). Favorise petites PRs.

0.7 5) Forking workflow (open-source)

- Chaque contributeur fork le repo principal, travaille dans son fork, ouvre Pull Request vers le repo parent.

Diagramme :

```
Upstream (org/repo)
  ^
  | Pull Request
Fork (user/repo) <- clone
```

Commandes :

```
# clone du fork
git clone git@github.com:utilisateur/repo.git
git remote add upstream git@github.com:org/repo.git
# synchroniser
git fetch upstream
git checkout main
git merge upstream/main
```

Avantages : sécurité (pas d'accès direct), isolation des contributeurs. Inconvénients : overhead pour synchroniser forks.

0.8 6) Trunk-based development (TBD)

- Tout le monde intègre fréquemment (plusieurs fois par jour) sur `main` (ou `trunk`), via petites branches très courtes.
- Favorise feature flags, CI rapide, releases continues.

Avantages : réduit divergence, facilite CI/CD. Inconvénients : nécessite tests solides, discipline et feature flags.

0.9 Comparatif rapide

- Centralisé : très simple, pas recommandé pour équipes collaboratives.

- Feature branch : bon compromis pour la plupart des équipes.
- GitHub Flow : léger, idéal pour déploiements continus.
- Gitflow : adapté aux équipes avec releases planifiées.
- Forking : idéal pour contributions open-source.
- Trunk-based : requis pour livraison continue à grande échelle.

0.10 Choix pragmatique

- Petite équipe / produit en évolution rapide : GitHub Flow ou Feature Branch.
- Projet Open Source : Forking + PR.
- Produit avec releases planifiées : Gitflow.
- Organisation visant CI/CD mature : Trunk-based + feature flags.

0.11 Intégration CI/CD

- Chaque push sur une branche active déclenche pipelines (tests, lint, builds).
- PRs doivent passer la pipeline avant merge.
- Protect branch rules : exiger revue, tests verts, pas de push direct.

0.12 Bonnes pratiques transversales

- PRs petites et ciblées
- Messages de commit clairs
- Rébase interactif pour nettoyer l'historique local si nécessaire
- Protéger branches critiques (main, develop)
- Utiliser templates de PR et checklist de revue

0.13 Exercices pratiques (atelier)

1) Exercice 1 — Feature Branch

- Init un repo, créez `main` et `feature/x`, faites 3 commits, poussez et mergez via PR (ou

simulate local merge). Vérifiez `git log --graph`.

2) Exercice 2 — Gitflow (simulé)

- Depuis `develop`, créez `feature/a`, terminez-la, ouvrez `release/1.0`, corrigez un bug, créez `hotfix/1.0.1` et mergez.

3) Exercice 3 — Fork & sync

- Forkez un petit repo public, clonez votre fork, ajoutez un remote `upstream`, synchronisez votre main puis proposez un PR.

0.14 Commandes utiles (résumé)

```
# branches
git checkout -b feature/x
git branch --all
# push
git push -u origin feature/x
# merge
git checkout main
git merge --no-ff feature/x
# rebase
git checkout feature/x
git fetch origin
git rebase origin/main
# sync fork
git remote add upstream <url>
git fetch upstream
git merge upstream/main
```

0.15 Pièges fréquents

- Rebaser une branche partagée publiquement
- Pousser sur `main` sans revue
- PRs trop larges
- Manquer de tests automatiques

0.16 Ressources et lecture

- GitHub Flow — <https://guides.github.com/introduction/flow/>

- Vincent Driessen Gitflow — <http://nvie.com/posts/a-successful-git-branching-model/>
- Trunk-based development — <https://trunkbaseddevelopment.com/>
- Pro Git — <https://git-scm.com/book/fr/v2>

0.17 Conclusion & prochaines étapes

- Choisissez un workflow adapté à votre équipe et formalisez-le dans le README du projet.
- Configurez règles de branche et CI pour l'automatisation.

??? note «Notes» Proposer aux étudiants d'installer un repo d'exemples pour pratiquer chaque workflow.

1 Fin

Questions ? Voulez-vous que je crée un dépôt de démonstration contenant des branches et scripts pour pratiquer automatiquement ces workflows ?