Programmation et projet encadré - L8TI005

Git : travailler en groupe

Crédits supports : Serge Fleury, Marine Wauquier

Pierre Magistry pierre.magistry@inalco.fr
Yoann Dupont yoann.dupont@sorbonne-nouvelle.fr

2024-2025

Université Sorbonne-Nouvelle
INALCO
Université Paris-Nanterre

Objectifs du jour

- Rappels sur Git + nouvelles commandes

Objectifs du jour

- Rappels sur Git + nouvelles commandes
  - commandes prévues pour travailler en groupe sur un projet

Objectifs du jour

- Rappels sur Git + nouvelles commandes
  - commandes prévues pour travailler en groupe sur un projet
  - entraînement sur LearningGitBranching

Objectifs du jour

- Rappels sur Git + nouvelles commandes
  - commandes prévues pour travailler en groupe sur un projet
  - entraînement sur LearningGitBranching
  - TP de mise en pratique sur GitLab

Objectifs du jour

- Rappels sur Git + nouvelles commandes
  - commandes prévues pour travailler en groupe sur un projet
  - entraînement sur LearningGitBranching
  - TP de mise en pratique sur GitLab
- Faire interagir bash et python (TP)

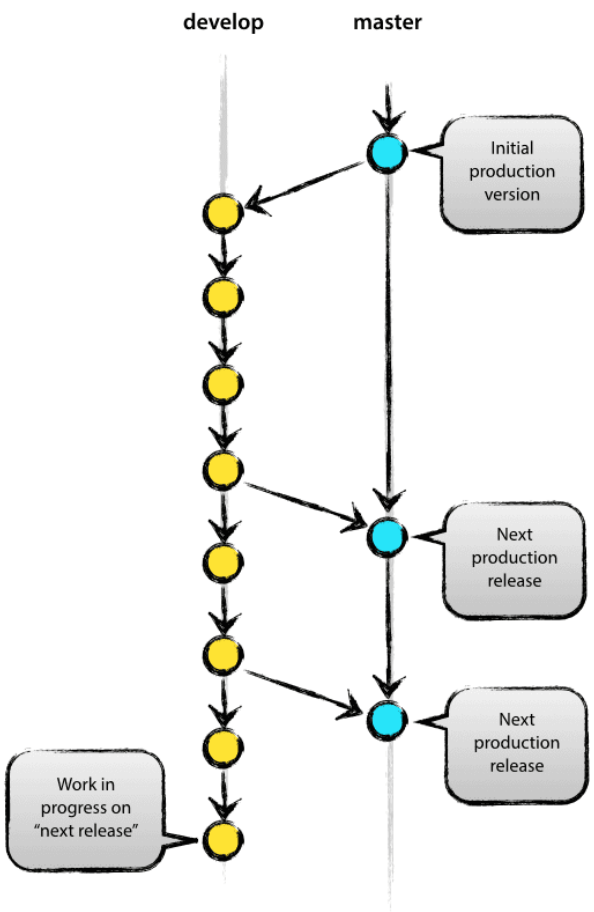
Git - Travailler en groupe

Fusil de Tchekov

« Supprimez tout ce qui n'est pas pertinent dans l'histoire. Si dans le premier acte vous dites qu'il y a un fusil accroché au mur, alors il faut absolument qu'un coup de feu soit tiré avec au second ou au troisième acte. S'il n'est pas destiné à être utilisé, il n'a rien à faire là. »
– Anton Tchekov

Fusil de Tchekov

« Supprimez tout ce qui n'est pas pertinent dans l'histoire. Si dans le premier acte vous dites qu'il y a un fusil accroché au mur, alors il faut absolument qu'un coup de feu soit tiré avec au second ou au troisième acte. S'il n'est pas destiné à être utilisé, il n'a rien à faire là. »
– Anton Tchekov

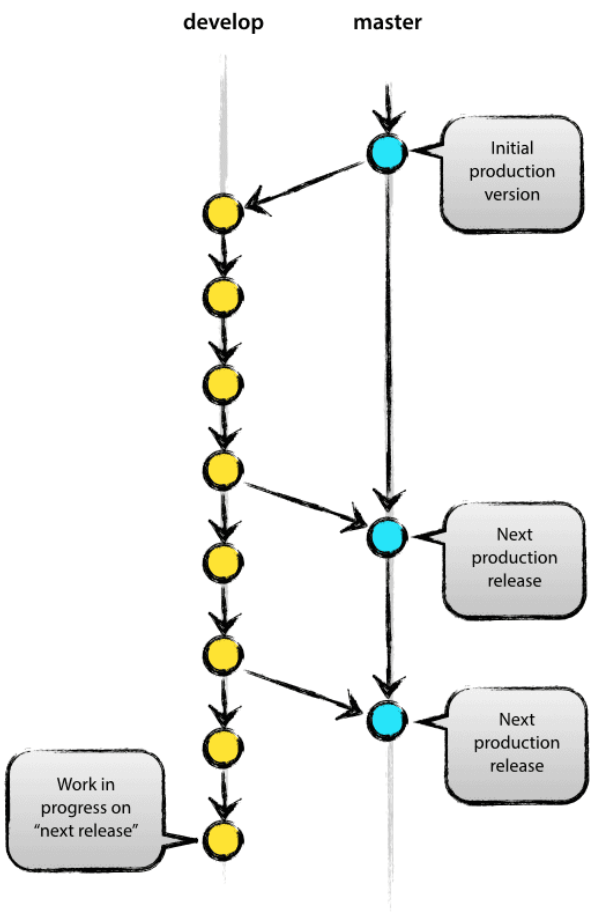


Vous vous souvenez de ce dessin du S1 ? Il illustre bien l'essence du travail en groupe sur git :

- Rappel : git ne voit que des modifications
- Un peu comme certains jeux vidéo : vos choix créent des branchements dans l'histoire, donc des déroulés alternatifs du récit

Fusil de Tchekov

« Supprimez tout ce qui n'est pas pertinent dans l'histoire. Si dans le premier acte vous dites qu'il y a un fusil accroché au mur, alors il faut absolument qu'un coup de feu soit tiré avec au second ou au troisième acte. S'il n'est pas destiné à être utilisé, il n'a rien à faire là. »
– Anton Tchekov

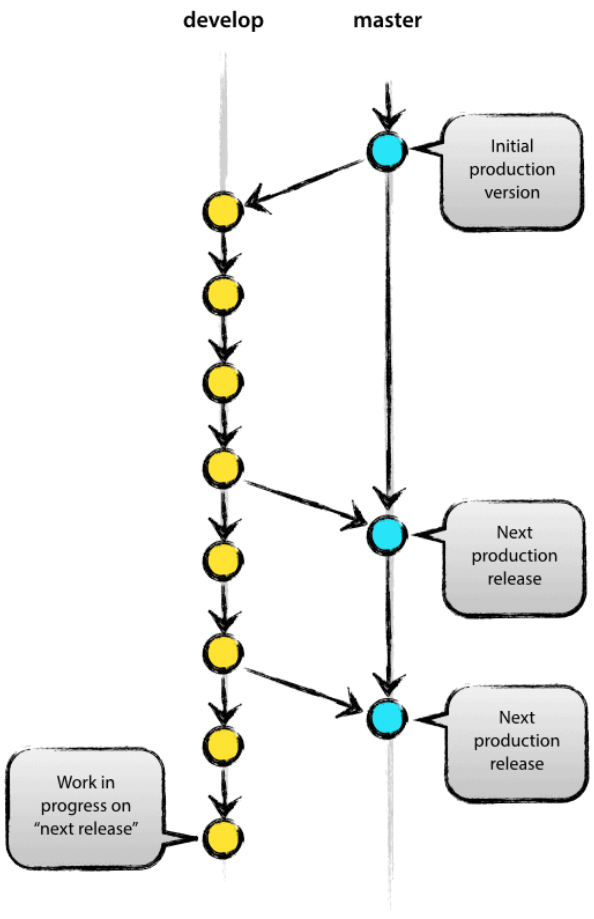


Vous vous souvenez de ce dessin du S1 ? Il illustre bien l'essence du travail en groupe sur git :

- Rappel : git ne voit que des modifications
- Un peu comme certains jeux vidéo : vos choix créent des branchements dans l'histoire, donc des déroulés alternatifs du récit
- Git, c'est un peu le même principe : on peut créer une histoire parallèle à notre dépôt (branche) qui n'affecte pas le reste

Fusil de Tchekov

« Supprimez tout ce qui n'est pas pertinent dans l'histoire. Si dans le premier acte vous dites qu'il y a un fusil accroché au mur, alors il faut absolument qu'un coup de feu soit tiré avec au second ou au troisième acte. S'il n'est pas destiné à être utilisé, il n'a rien à faire là. »
– Anton Tchekov



Vous vous souvenez de ce dessin du S1 ? Il illustre bien l'essence du travail en groupe sur git :

- Rappel : git ne voit que des modifications
- Un peu comme certains jeux vidéo : vos choix créent des branchements dans l'histoire, donc des déroulés alternatifs du récit
- Git, c'est un peu le même principe : on peut créer une histoire parallèle à notre dépôt (branche) qui n'affecte pas le reste
- On peut récupérer des modifications faites ailleurs pour les intégrer à notre branche. Il y a deux stratégies : *merge* et *rebase*.

D'où on part et où on va

- Au premier semestre : travail en groupe $\rightarrow$ on se coordonne pour pas commit/push en même temps.
  - Ça marche uniquement en petits groupes quand on a un moyen de communication instantané
  - Le travail est au final plutôt synchrone, un peu contraire à la philosophie git

D'où on part et où on va

- Au premier semestre : travail en groupe $\rightarrow$ on se coordonne pour pas commit/push en même temps.
  - Ça marche uniquement en petits groupes quand on a un moyen de communication instantané
  - Le travail est au final plutôt synchrone, un peu contraire à la philosophie git
- Maintenant : Chaque personne va faire une partie du travail de son côté et tout sera regroupé à la fin
  - Travail vraiment asynchrone : personne ne voit ce que fait les autres, mais on ne se marche pas dessus
  - Permet une division des tâches plus claire et simple, facilite le travail sur des versions spécifiques, la correction de bug, etc.
  - Plus puissant, mais plus compliqué : il faut tout regrouper à la fin, et les conflits doivent être gérés manuellement

D'où on part et où on va

- Au premier semestre : travail en groupe $\rightarrow$ on se coordonne pour pas commit/push en même temps.
  - Ça marche uniquement en petits groupes quand on a un moyen de communication instantané
  - Le travail est au final plutôt synchrone, un peu contraire à la philosophie git
- Maintenant : Chaque personne va faire une partie du travail de son côté et tout sera regroupé à la fin
  - Travail vraiment asynchrone : personne ne voit ce que fait les autres, mais on ne se marche pas dessus
  - Permet une division des tâches plus claire et simple, facilite le travail sur des versions spécifiques, la correction de bug, etc.
  - Plus puissant, mais plus compliqué : il faut tout regrouper à la fin, et les conflits doivent être gérés manuellement
- Besoin d'ajouter des commandes à notre arsenal ou des options à d'anciennes commandes

Gérer les branchements de votre dépôt : git branch

Permet de gérer les branches d'un projet.

Une branche est une dérivation dans l'historique de git à partir d'un point nommé.
Les branches permettent de travailler dans une version alternative de votre dépôt.

```
git branch [-options...]
```

Documentation : https://git-scm.com/docs/git-branch

Gérer les branchements de votre dépôt : git branch

Permet de gérer les branches d'un projet.

Une branche est une dérivation dans l'historique de git à partir d'un point nommé. Les branches permettent de travailler dans une version alternative de votre dépôt.

```
git branch [-options...]
```

Quelques usages :

- *--list* ou rien : liste les branches
- `git branch <nom-branche> [reference]` : créer la branche <nom-branche> depuis [reference]. [reference] → HEAD par défaut
- `git branch -m/-c [ancien] <nouveau>` : renommer/copier une branche
- `git branch -d <nom>` : supprimer une branche

Documentation : https://git-scm.com/docs/git-branch

Changer de branche : git checkout et git switch (1/2)

Deux alternatives : `git checkout` ou `git switch`. Vous trouverez plus souvent checkout dans la vraie vie^{TM}, switch est expérimental mais voué à se stabiliser.

Documentation : https://git-scm.com/docs/git-checkout et https://git-scm.com/docs/git-switch

Changer de branche : git checkout et git switch (1/2)

Deux alternatives : `git checkout` ou `git switch`. Vous trouverez plus souvent checkout dans la vraie vie^{TM}, switch est expérimental mais voué à se stabiliser.

```
git checkout [-b] <branche>
```

Bascule sur la branche <branche>. L'option `-b` permet de la créer au passage.

Documentation : https://git-scm.com/docs/git-checkout et https://git-scm.com/docs/git-switch

Changer de branche : git checkout et git switch (1/2)

Deux alternatives : `git checkout` ou `git switch`. Vous trouverez plus souvent checkout dans la vraie vie^{TM}, switch est expérimental mais voué à se stabiliser.

```
git checkout [-b] <branche>
```

Bascule sur la branche <branche>. L'option `-b` permet de la créer au passage.

Si on veut basculer sur la branche `MyBranch` qui existe sur le dépôt distant (l'origine du dépôt), on indiquera typiquement :
`git checkout origin/MyBranch`.

Documentation : https://git-scm.com/docs/git-checkout et https://git-scm.com/docs/git-switch

Changer de branche : git checkout et git switch (1/2)

Deux alternatives : `git checkout` ou `git switch`. Vous trouverez plus souvent checkout dans la vraie vie^{TM}, switch est expérimental mais voué à se stabiliser.

```
git checkout [-b] <branche>
```

Bascule sur la branche <branche>. L'option `-b` permet de la créer au passage.

Si on veut basculer sur la branche `MyBranch` qui existe sur le dépôt distant (l'origine du dépôt), on indiquera typiquement :
`git checkout origin/MyBranch`.

Pour éviter ce genre de problème, on peut utiliser `git switch MyBranch`, qui va chercher parmi toutes les branches celle qui porte le nom `MyBranch`.

Documentation : https://git-scm.com/docs/git-checkout et https://git-scm.com/docs/git-switch

Changer de branche : git checkout et git switch (1/2)

Deux alternatives : `git checkout` ou `git switch`. Vous trouverez plus souvent checkout dans la vraie vie^{TM}, switch est expérimental mais voué à se stabiliser.

```
git checkout [-b] <branche>
```

Bascule sur la branche <branche>. L'option `-b` permet de la créer au passage.

Si on veut basculer sur la branche `MyBranch` qui existe sur le dépôt distant (l'origine du dépôt), on indiquera typiquement :
`git checkout origin/MyBranch`.

Pour éviter ce genre de problème, on peut utiliser `git switch MyBranch`, qui va chercher parmi toutes les branches celle qui porte le nom `MyBranch`.

Pensez à fetch avant !

Documentation : https://git-scm.com/docs/git-checkout et https://git-scm.com/docs/git-switch

Changer de branche : git checkout et git switch (2/2)

`git checkout` étant une commande très compliquée, deux commandes ont été proposée pour gérer deux comportements. Pour changer de branche : `git switch`.[1]

Documentation : https://git-scm.com/docs/git-checkout et https://git-scm.com/docs/git-switch

[1] Attention, cette commande est encore expérimentale.

Changer de branche : git checkout et git switch (2/2)

`git checkout` étant une commande très compliquée, deux commandes ont été proposée pour gérer deux comportements. Pour changer de branche : `git switch`.[1]

```
git switch [-c] <branche>
```

Bascule sur la branche <branche>. L'option `-c` permet de la créer au passage.

Documentation : https://git-scm.com/docs/git-checkout et https://git-scm.com/docs/git-switch

[1] Attention, cette commande est encore expérimentale.

Changer de branche : git checkout et git switch (2/2)

`git checkout` étant une commande très compliquée, deux commandes ont été proposée pour gérer deux comportements. Pour changer de branche : `git switch`.[1]

```
git switch [-c] <branche>
```

Bascule sur la branche <branche>. L'option `-c` permet de la créer au passage.

Si on veut basculer sur la branche `MyBranch` qui existe sur le dépôt distant (l'origine du dépôt), `git switch` se débrouille pour trouver la bonne branche.

Documentation : https://git-scm.com/docs/git-checkout et https://git-scm.com/docs/git-switch

[1] Attention, cette commande est encore expérimentale.

Raccorder une branche au dépôt distant : git push

Quand on crée une branche depuis son terminal, la branche créée est locale. Elle n'existe que sur votre ordinateur.

Documentation : https://git-scm.com/docs/git-push et https://git-scm.com/docs/git-switch

Raccorder une branche au dépôt distant : git push

Quand on crée une branche depuis son terminal, la branche créée est locale. Elle n'existe que sur votre ordinateur.

Pour la relier au dépôt distant, le **premier** push depuis une branche doit inclure :

```
git push --set-upstream origin <branche>
```

Documentation : https://git-scm.com/docs/git-push et https://git-scm.com/docs/git-switch

Raccorder une branche au dépôt distant : git push

Quand on crée une branche depuis son terminal, la branche créée est locale. Elle n'existe que sur votre ordinateur.

Pour la relier au dépôt distant, le **premier** push depuis une branche doit inclure :

```
git push --set-upstream origin <branche>
```

Où <branche> est typiquement le nom de la branche courante.

Documentation : https://git-scm.com/docs/git-push et https://git-scm.com/docs/git-switch

Récupérer des changements d'ailleurs v1 : git merge

Permet de récupérer dans l'état courant (HEAD) les modifications faites jusqu'à un autre *commit* depuis leur point de divergence.
De base, *merge* rattache les modifications *commit* à HEAD. Il crée alors un nouveau commit (commit de *merge*), qui indique que la fusion a eu lieu. Le commit de *merge* a donc deux parents.

```
git merge [-options...]  <commit>
```

Documentation : https://git-scm.com/docs/git-merge

Récupérer des changements d'ailleurs v1 : git merge

Permet de récupérer dans l'état courant (HEAD) les modifications faites jusqu'à un autre *commit* depuis leur point de divergence.
De base, *merge* rattache les modifications *commit* à HEAD. Il crée alors un nouveau commit (commit de *merge*), qui indique que la fusion a eu lieu. Le commit de *merge* a donc deux parents.

```
git merge [-options...]  <commit>
```

<commit> est toute façon d'accéder à un commit : hash, tag, nom de branche...

Documentation : https://git-scm.com/docs/git-merge

Récupérer des changements d'ailleurs v1 : git merge

Permet de récupérer dans l'état courant (HEAD) les modifications faites jusqu'à un autre *commit* depuis leur point de divergence.
De base, *merge* rattache les modifications *commit* à HEAD. Il crée alors un nouveau commit (commit de *merge*), qui indique que la fusion a eu lieu. Le commit de *merge* a donc deux parents.

```
git merge [-options...]  <commit>
```

<commit> est toute façon d'accéder à un commit : hash, tag, nom de branche...

Quelques usages :

- `git merge <commit>` : récupère l'historique de <commit>
- `git merge --squash <commit>` : récupère l'historique de <commit> et "écrase" l'historique comme étant un seul commit

Documentation : https://git-scm.com/docs/git-merge

Récupérer des changements d'ailleurs v2 : git rebase

Analogue à *merge*, mais gère l'historique différemment. *rebase* rejoue tous les commits à partir du point d'origine. Il fait donc comme si on avait continué de travailler sans branche. Contrairement à *merge*, on garde donc un parent unique.

```
git rebase <commit>
```

Documentation : https://git-scm.com/docs/git-rebase

Récupérer des changements d'ailleurs v2 : git rebase

Analogue à *merge*, mais gère l'historique différemment. *rebase* rejoue tous les commits à partir du point d'origine. Il fait donc comme si on avait continué de travailler sans branche. Contrairement à *merge*, on garde donc un parent unique.

```
git rebase <commit>
```

<commit> est toute façon d'accéder à un commit : hash, tag, nom de branche...

Documentation : https://git-scm.com/docs/git-rebase

Récupérer des changements d'ailleurs v2 : git rebase

Analogue à *merge*, mais gère l'historique différemment. *rebase* rejoue tous les commits à partir du point d'origine. Il fait donc comme si on avait continué de travailler sans branche. Contrairement à *merge*, on garde donc un parent unique.

```
git rebase <commit>
```

<commit> est toute façon d'accéder à un commit : hash, tag, nom de branche...

Quelques usages :

- `git rebase <branche>` : rejoue les commit de <branche> à partir de HEAD

Documentation : https://git-scm.com/docs/git-rebase

Exercices

- Faire les exercices donnés sur icampus sur le site *LearnGitBranching*
- Faire le TP git
  - créer le tag "seance1" dans la branche `main` à la fin du travail
- Date limite de rendu : dimanche 2 février à 23h59