Rédaction mathématique et informatique

Cette semaine

git :

- Rappels et retours
- Collaboration
- Dépôt distant
- Fusion de version

LaTeX :

- Retour sur les tableaux
- Inclure du contenu LaTeX depuis un autre fichier
- Insérer des images
- Définir ses propres commandes

Retours sur git

Utilisation standard (seul)

Dans un dépôt git,

Utilisation standard (seul)

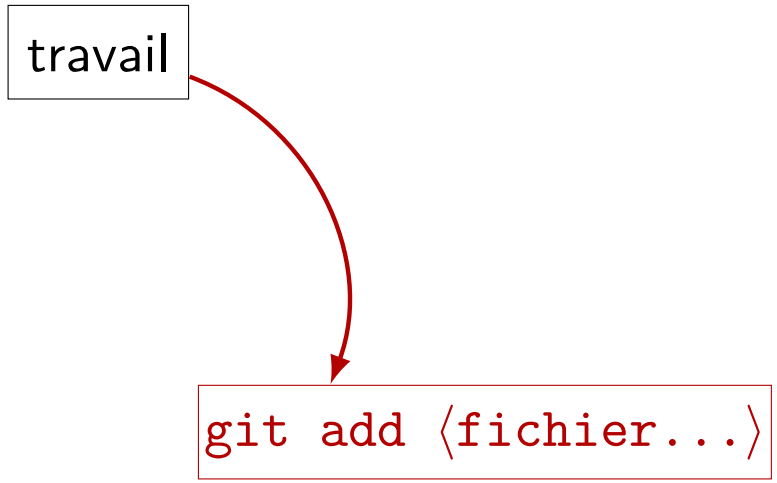
Dans un dépôt git,

travail

1. On travaille sur nos fichiers $\Longrightarrow$ modifications.

Utilisation standard (seul)

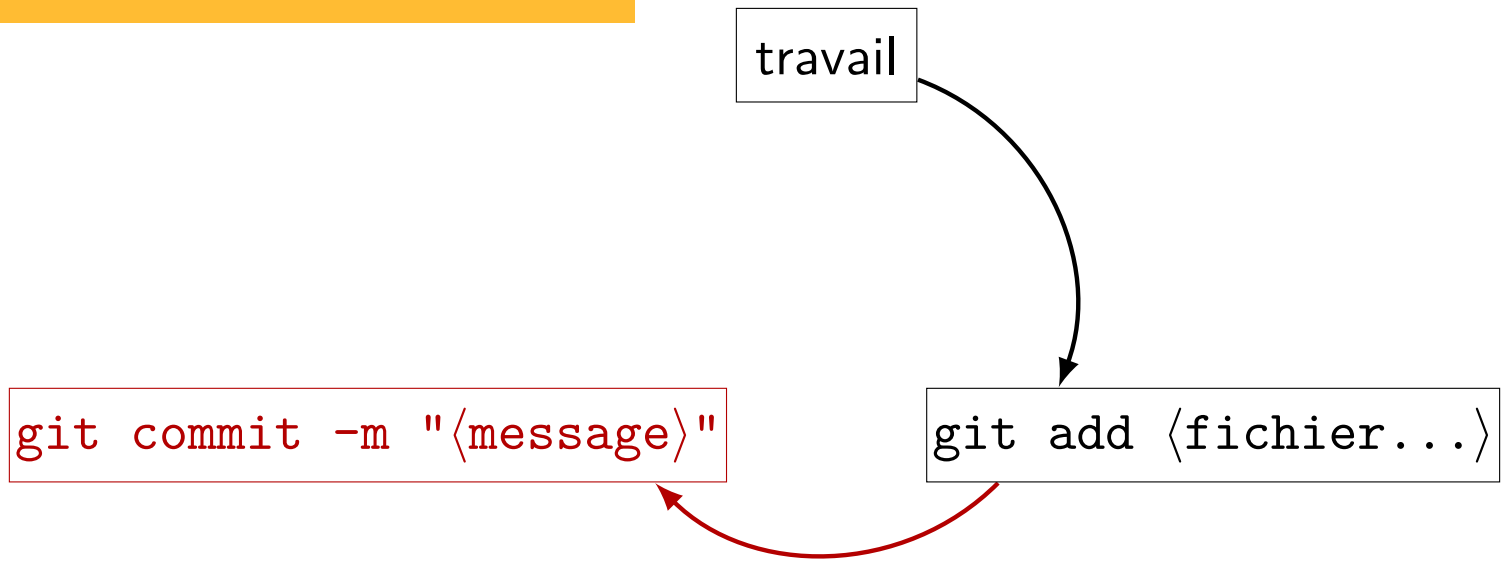
Dans un dépôt git,



1. On travaille sur nos fichiers $\Longrightarrow$ modifications.
2. On montre à git nos changements.

Utilisation standard (seul)

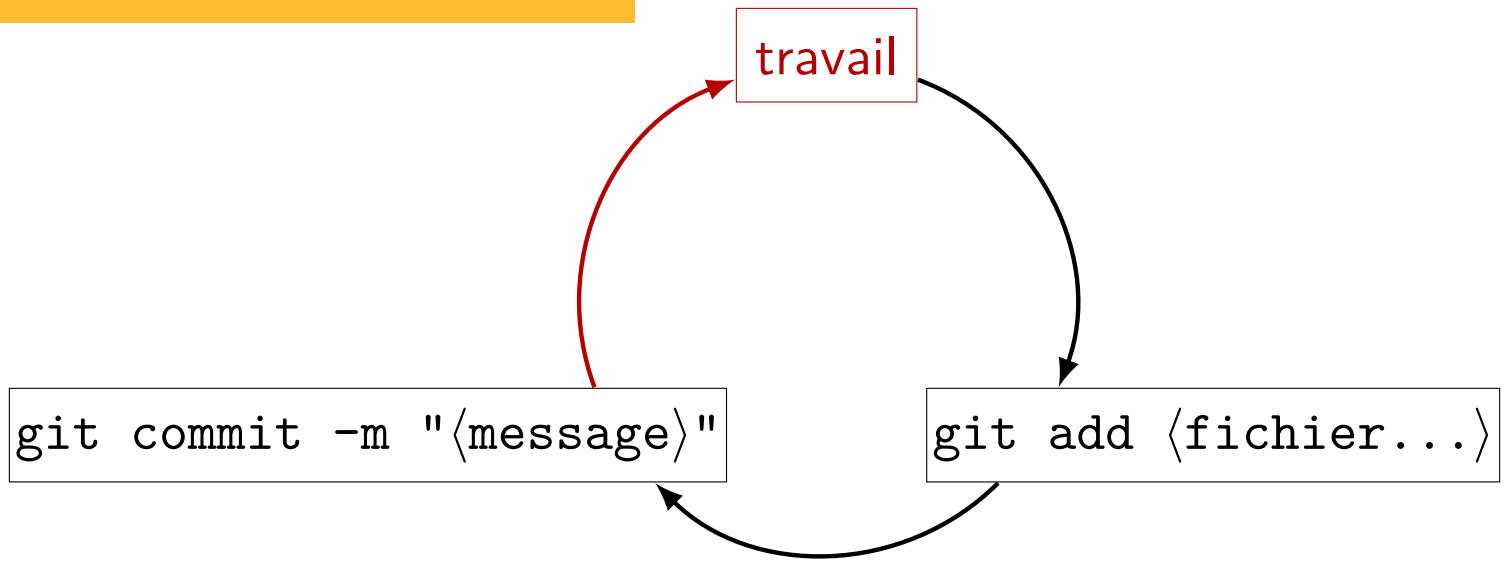
Dans un dépôt git,



1. On travaille sur nos fichiers $\Longrightarrow$ modifications.
2. On montre à git nos changements.
3. On demande à git de faire une version avec message.

Utilisation standard (seul)

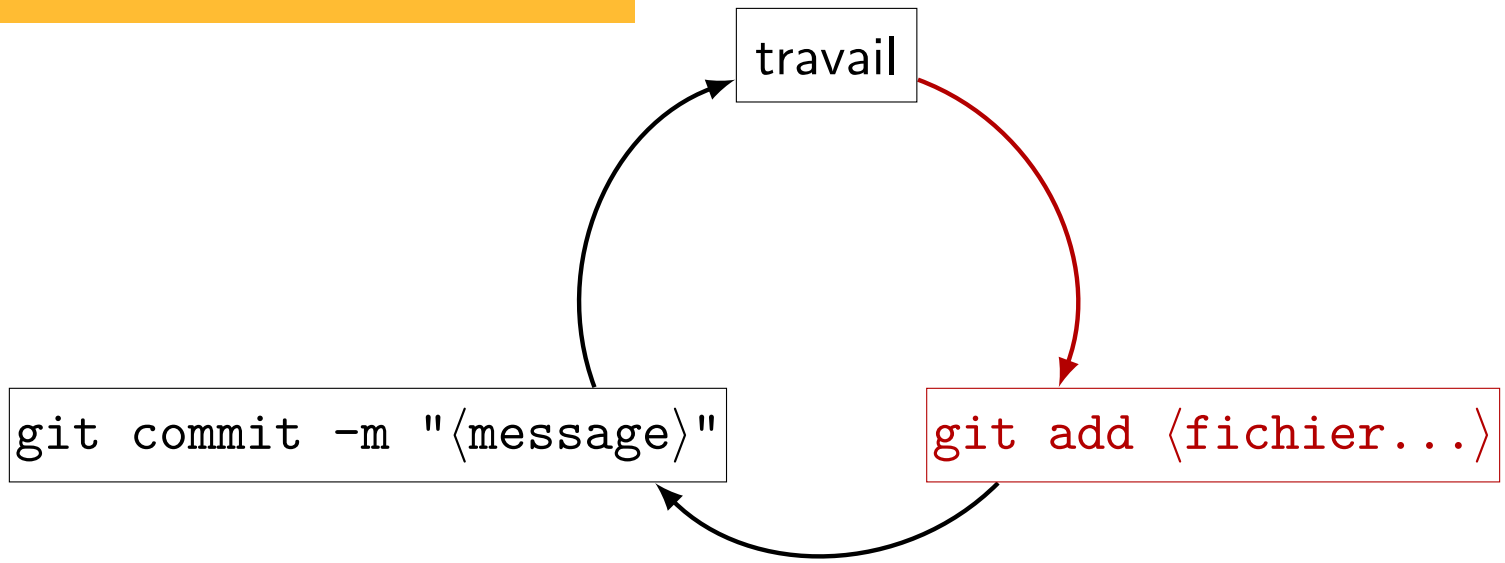
Dans un dépôt git,



1. On travaille sur nos fichiers $\Longrightarrow$ modifications.
2. On montre à git nos changements.
3. On demande à git de faire une version avec message.

Utilisation standard (seul)

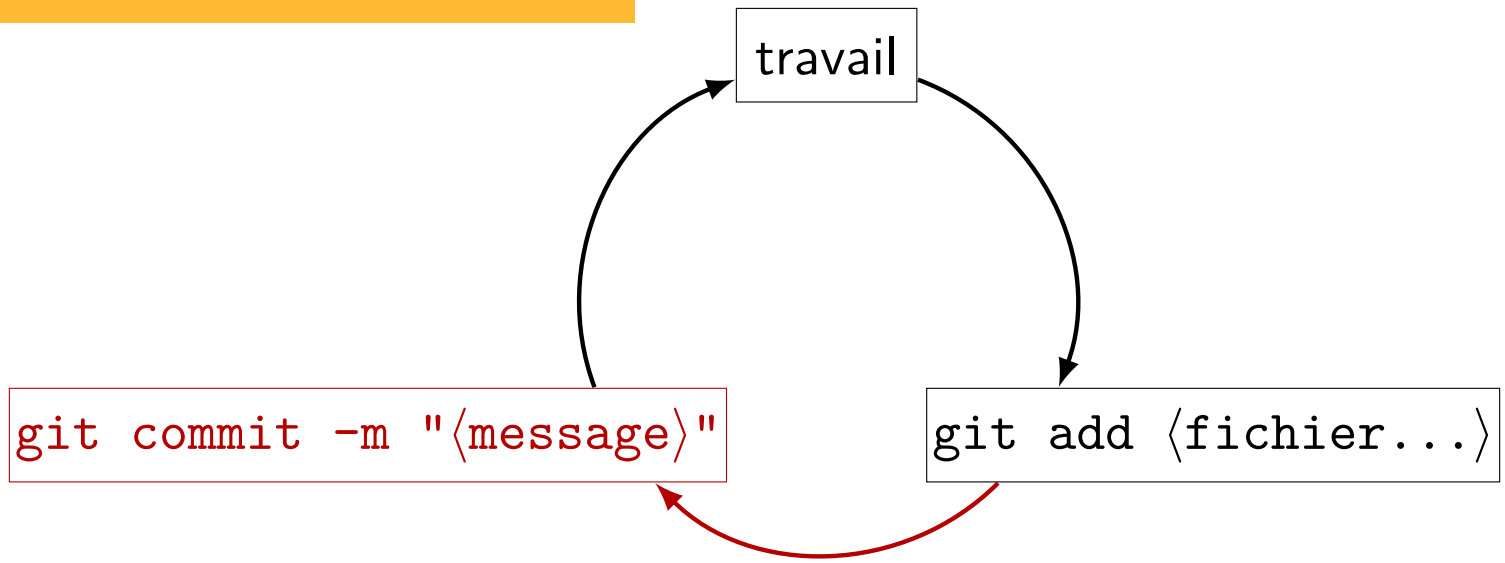
Dans un dépôt git,



1. On travaille sur nos fichiers $\Longrightarrow$ modifications.
2. On montre à git nos changements.
3. On demande à git de faire une version avec message.

Utilisation standard (seul)

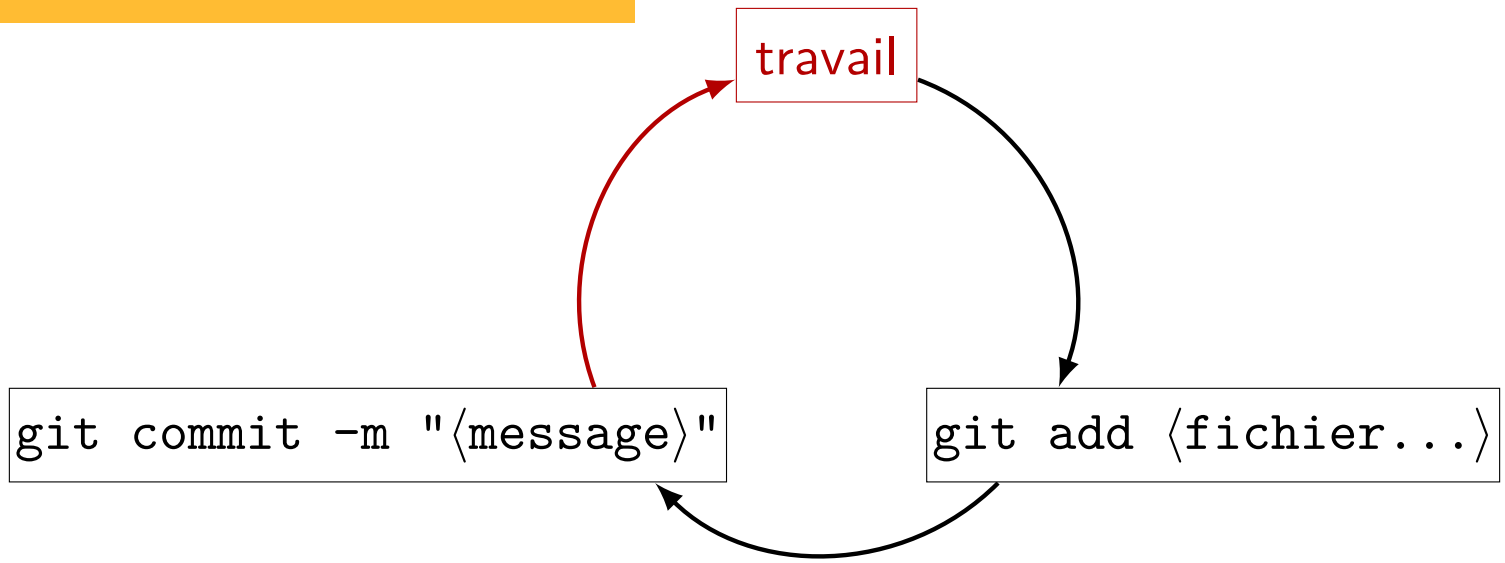
Dans un dépôt git,



1. On travaille sur nos fichiers $\Longrightarrow$ modifications.
2. On montre à git nos changements.
3. On demande à git de faire une version avec message.

Utilisation standard (seul)

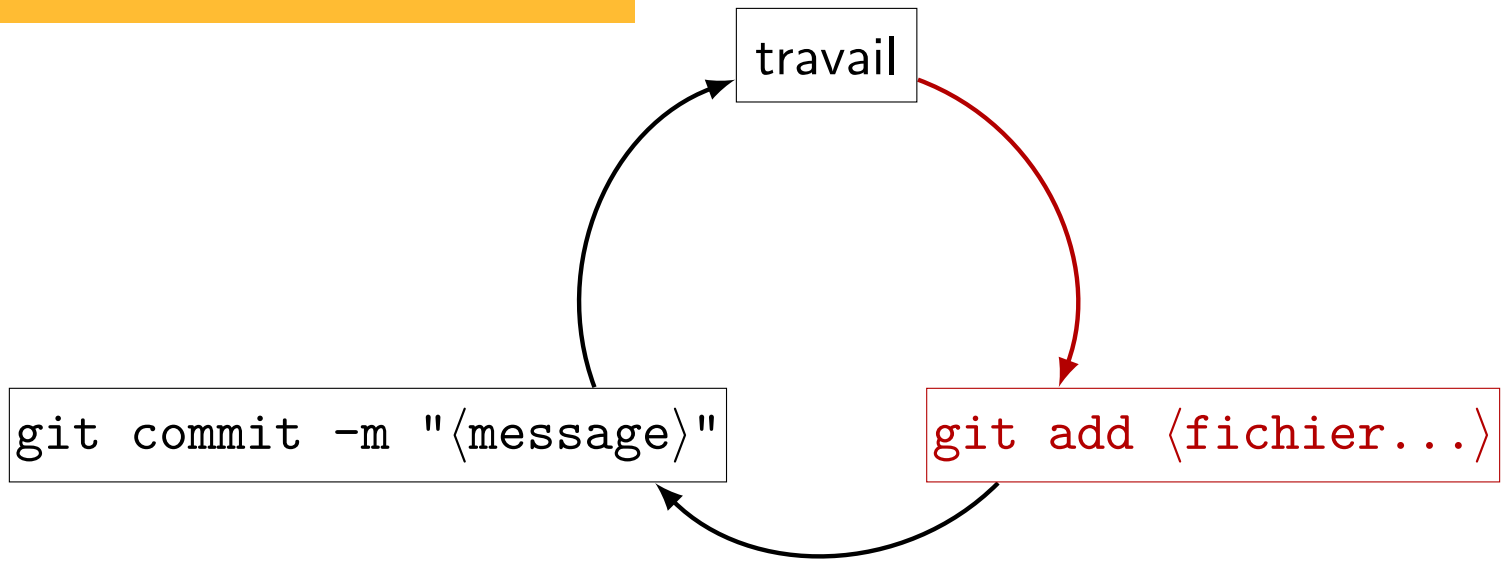
Dans un dépôt git,



1. On travaille sur nos fichiers $\Longrightarrow$ modifications.
2. On montre à git nos changements.
3. On demande à git de faire une version avec message.

Utilisation standard (seul)

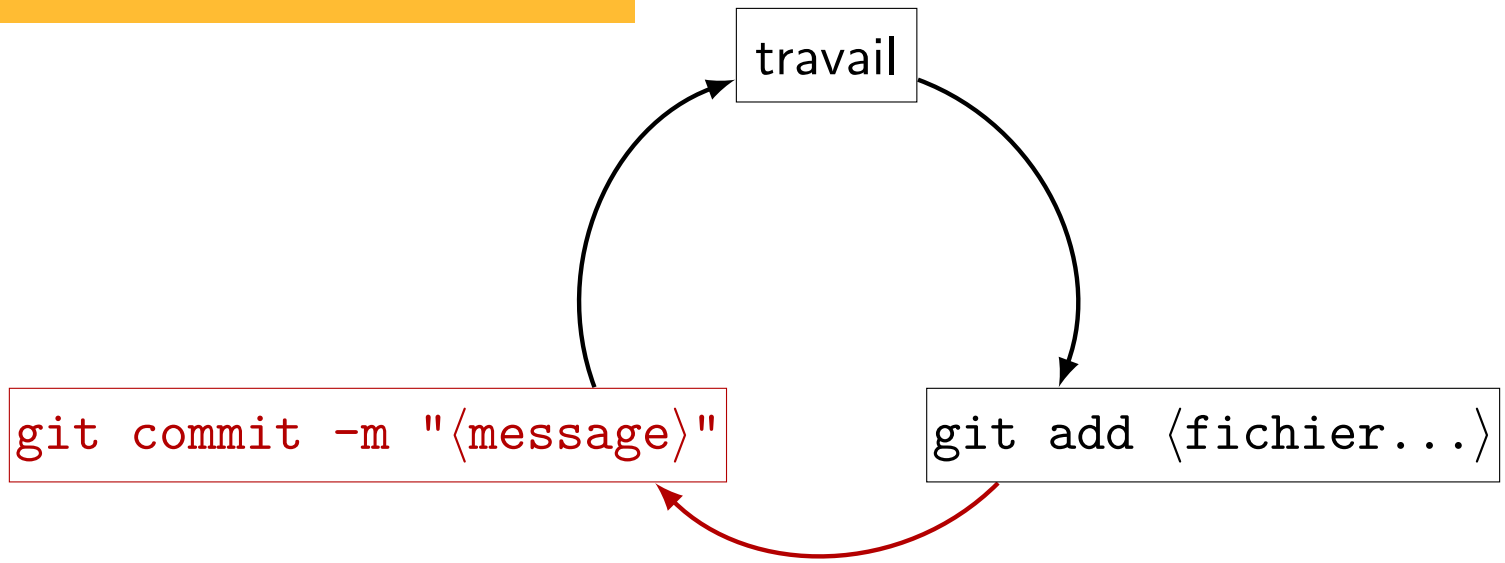
Dans un dépôt git,



1. On travaille sur nos fichiers $\Longrightarrow$ modifications.
2. On montre à git nos changements.
3. On demande à git de faire une version avec message.

Utilisation standard (seul)

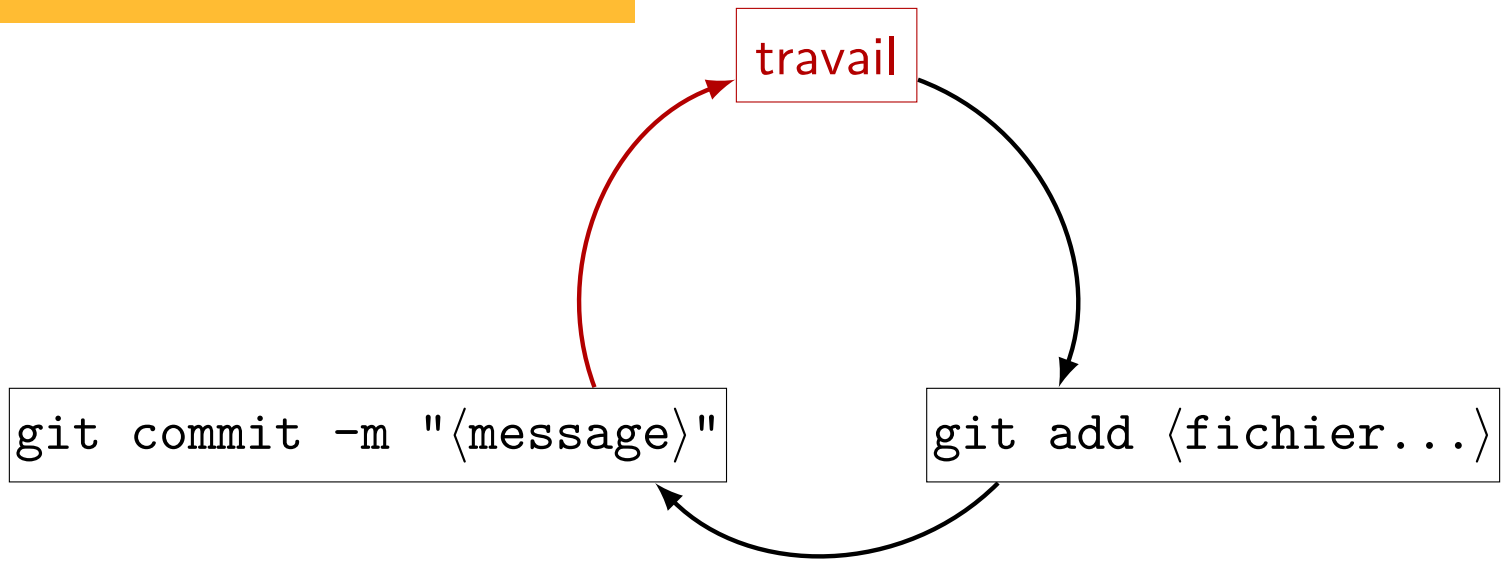
Dans un dépôt git,



1. On travaille sur nos fichiers $\Longrightarrow$ modifications.
2. On montre à git nos changements.
3. On demande à git de faire une version avec message.

Utilisation standard (seul)

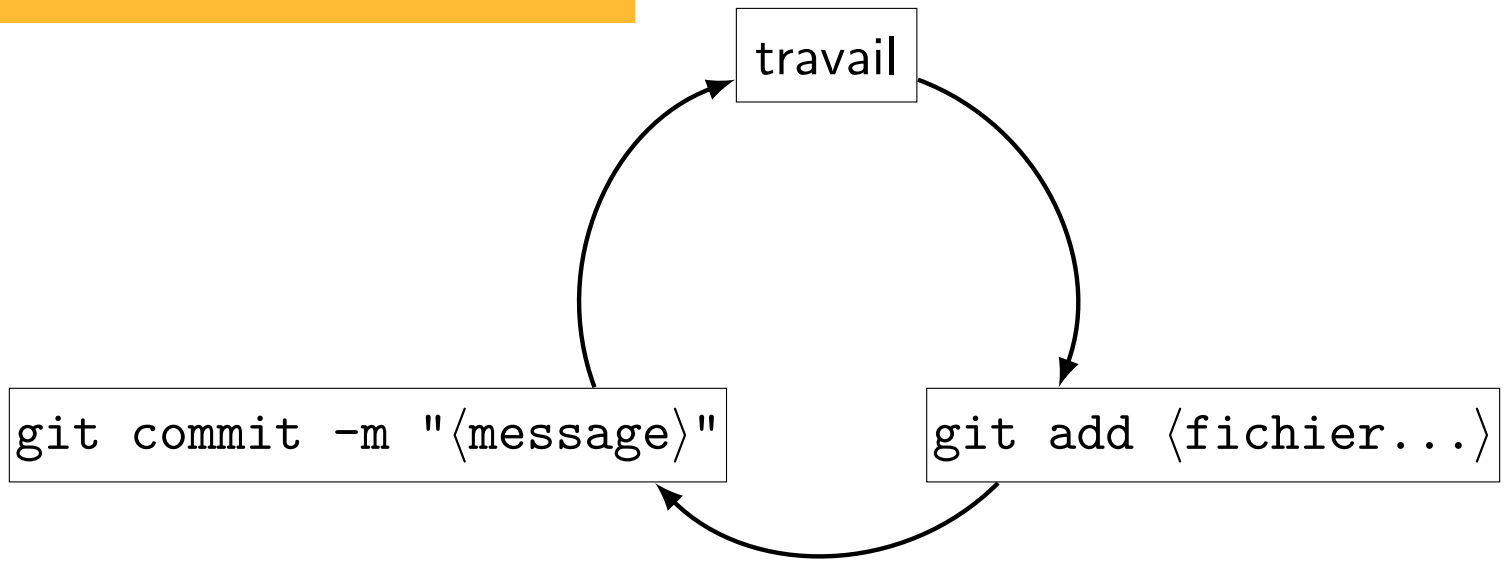
Dans un dépôt git,



1. On travaille sur nos fichiers $\Longrightarrow$ modifications.
2. On montre à git nos changements.
3. On demande à git de faire une version avec message.

Utilisation standard (seul)

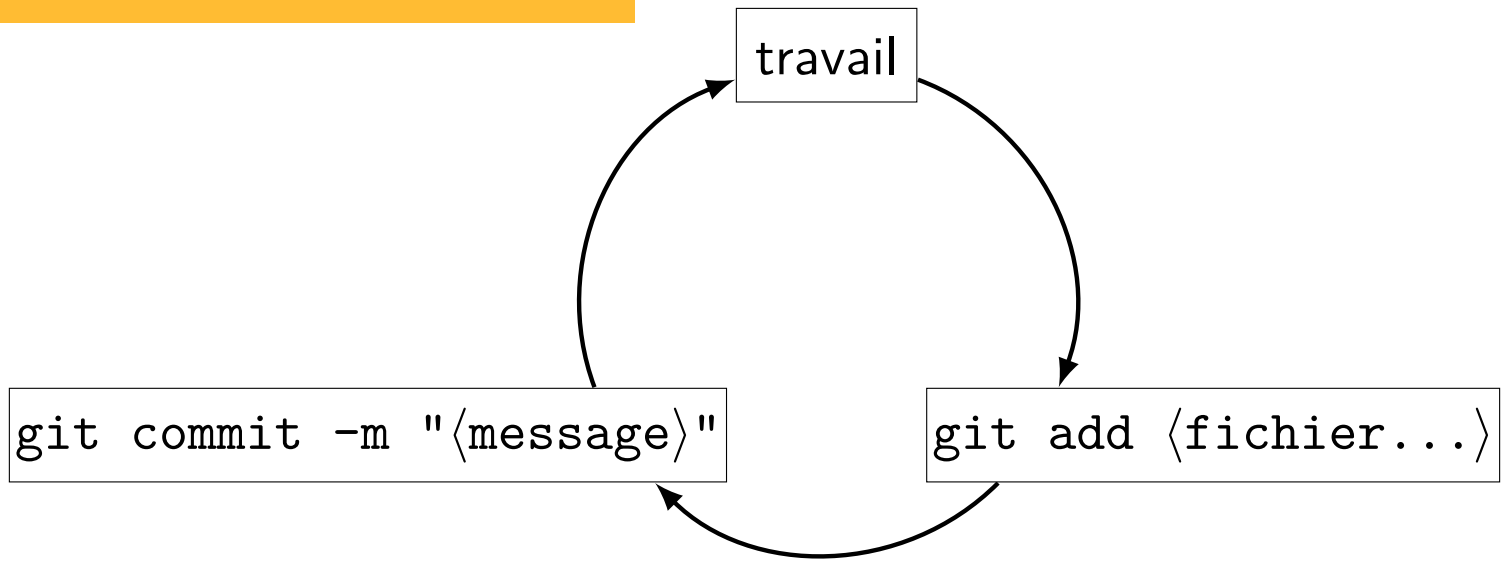
Dans un dépôt git,



1. On travaille sur nos fichiers $\Longrightarrow$ modifications.
2. On montre à git nos changements.
3. On demande à git de faire une version avec message.

Utilisation standard (seul)

Dans un dépôt git,

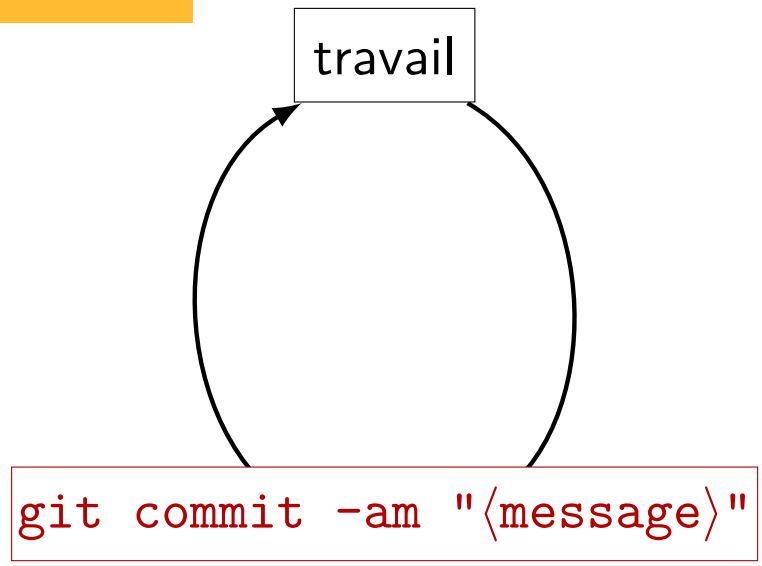


1. On travaille sur nos fichiers $\Longrightarrow$ modifications.
2. On montre à git nos changements.
3. On demande à git de faire une version avec message.

Deux raccourcis, avec `git commit` :

Utilisation standard (seul)

Dans un dépôt git,



1. On travaille sur nos fichiers $\Longrightarrow$ modifications.
2. On montre à git nos changements.
3. On demande à git de faire une version avec message.

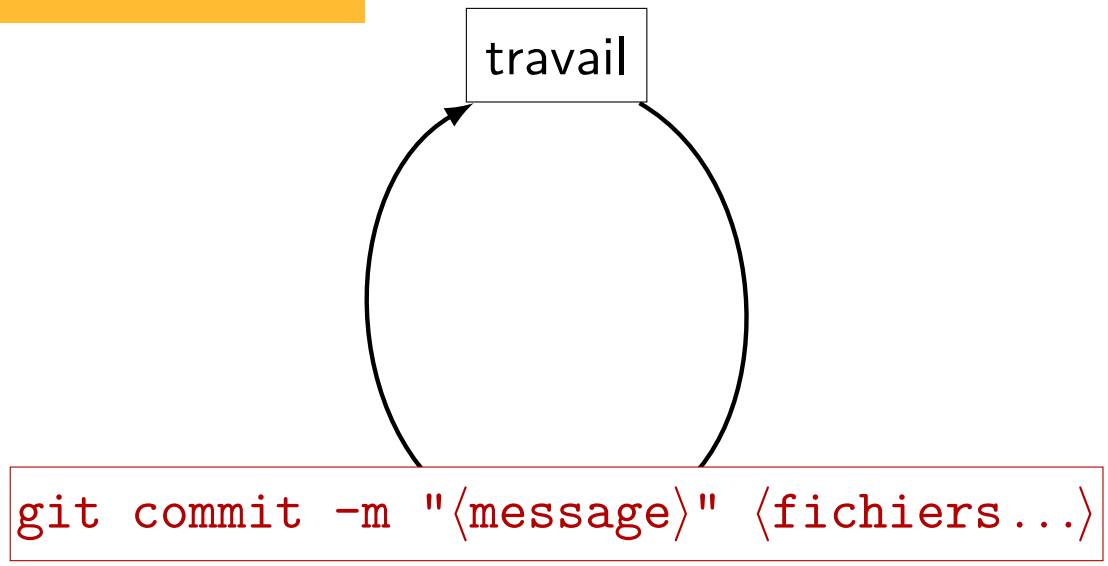
Deux raccourcis, avec `git commit` :

- L'option `-a`/`--all` :

  $\Longrightarrow$ ajoute tous les fichiers connus de git à la version.

Utilisation standard (seul)

Dans un dépôt git,



1. On travaille sur nos fichiers $\Longrightarrow$ modifications.
2. On montre à git nos changements.
3. On demande à git de faire une version avec message.

Deux raccourcis, avec `git commit` :

- L'option `-a`/`--all` :
  $\Longrightarrow$ ajoute tous les fichiers connus de git à la version.
- Les arguments ⟨`fichiers...`⟩ :
  $\Longrightarrow$ n'inclut dans la version que les fichiers connus de git indiqués.

Que mettre dans un `commit` ?

1 version = 1 changement

Que mettre dans un `commit` ?

1 version = 1 unité de changement

Que mettre dans un `commit` ?

1 version = 1 unité de changement

- Faire des versions très régulièrement,
  sans attendre la fin de la session de travail ;
- Découper un `commit` en plusieurs
  (et donc, ne pas abuser de l'option `-a` ou de `git add ./`) ;

Que mettre dans un `commit` ?

1 version = 1 unité de changement

- Faire des versions très régulièrement,
  sans attendre la fin de la session de travail ;
- Découper un `commit` en plusieurs
  (et donc, ne pas abuser de l'option `-a` ou de `git add ./`) ;

- Donner un message informatif sur la nature du changement.

Quels fichiers suivre avec git ?

- fichiers sources (*e.g.*, `.tex`, `.py`, `.c`) ;
- les fichiers `.gitignore` ;
- quelques fichiers objets (*e.g.*, images) si utilisés par le source.

Quels fichiers suivre avec git ?

- fichiers sources (*e.g.*, `.tex`, `.py`, `.c`) ;
- les fichiers `.gitignore` ;
- quelques fichiers objets (*e.g.*, images) si utilisés par le source.

Le fichier `.gitignore`

Sert à indiquer à git qu'il doit ignorer certains fichiers du répertoire. Ainsi :

- la sortie de `git status` reste propre ;
- les fichiers inutiles ne sont pas ajoutés par erreur (*e.g.*, *via* `git add ./`).

Quels fichiers suivre avec git ?

- fichiers sources (*e.g.*, `.tex`, `.py`, `.c`) ;
- les fichiers `.gitignore` ;
- quelques fichiers objets (*e.g.*, images) si utilisés par le source.

Le fichier `.gitignore`

Sert à indiquer à git qu'il doit ignorer certains fichiers du répertoire. Ainsi :

- la sortie de `git status` reste propre ;
- les fichiers inutiles ne sont pas ajoutés par erreur (*e.g.*, *via* `git add ./`).

Typiquement, on ignorera :

- des fichiers objets (*e.g.*, `.pdf`, `.o`) ;
- des fichiers auxiliaires (*e.g.*, `.aux`, `.log`, *etc...*) ;
- des sous répertoires (*e.g.*, `./tmp/`, `./docs/`).

Configuration git

git a besoin de quelques informations.

Configuration git

git a besoin de quelques informations.

on peut les indiquer avec `git config`.

Configuration git

git a besoin de quelques informations.

on peut les indiquer avec `git config`.

Les plus indispensables sont :

- l'identité de l'utilisateur
- l'*email* de l'utilisateur

(et vous n'aurez pas besoin de voir tellement plus).

Configuration git

git a besoin de quelques informations.

on peut les indiquer avec `git config`.

Les plus indispensables sont :

- l'identité de l'utilisateur

```
git config --global user.name "⟨Prénom NOM DE FAMILLE⟩"
```

- l'*email* de l'utilisateur

```
git config --global user.email "⟨identite@domain.ext⟩"
```

(et vous n'aurez pas besoin de voir tellement plus).

Configuration git

git a besoin de quelques informations.

on peut les indiquer avec `git config`.

Les plus indispensables sont :

- l'identité de l'utilisateur

```
git config --global user.name "⟨Prénom NOM DE FAMILLE⟩"
```

- l'*email* de l'utilisateur

```
git config --global user.email "⟨identite@domain.ext⟩"
```

(et vous n'aurez pas besoin de voir tellement plus).

Normalement `--global` configure git pour notre (utilisateur) système.

Configuration git

git a besoin de quelques informations.

on peut les indiquer avec `git config`.

Les plus indispensables sont :

- l'identité de l'utilisateur

```
git config --global user.name "⟨Prénom NOM DE FAMILLE⟩"
```

- l'*email* de l'utilisateur

```
git config --global user.email "⟨identite@domain.ext⟩"
```

(et vous n'aurez pas besoin de voir tellement plus).

Normalement `--global` configure git pour notre (utilisateur) système.

Mais les sessions sur les machines du SCI ne sont pas persistantes

et la configuration est donc perdue.

Configuration git

git a besoin de quelques informations.

on peut les indiquer avec `git config`.

Les plus indispensables sont :

- l'identité de l'utilisateur

```
git config --global user.name "⟨Prénom NOM DE FAMILLE⟩"
```

- l'*email* de l'utilisateur

```
git config --global user.email "⟨identite@domain.ext⟩"
```

(et vous n'aurez pas besoin de voir tellement plus).

Normalement `--global` configure git pour notre (utilisateur) système.

Mais les sessions sur les machines du SCI ne sont pas persistantes

et la configuration est donc perdue.

Donc retenez bien les deux commandes ci-dessus

car vous devrez les refaire.

La collaboration avec git

Système distribué

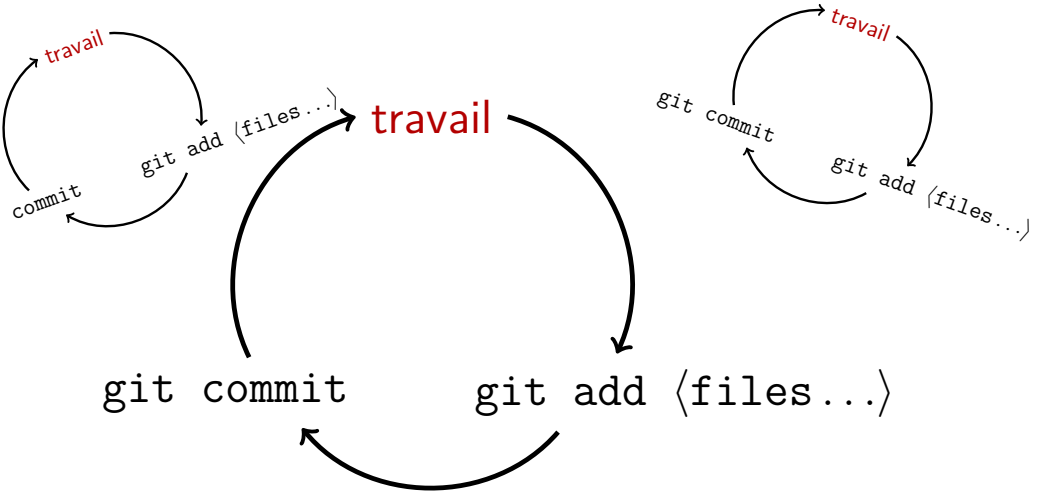
Chaque collaborateur a sa propre “copie” du dépôt git :

on parle de clone.

Système distribué

Chaque collaborateur a sa propre “copie” du dépôt git :

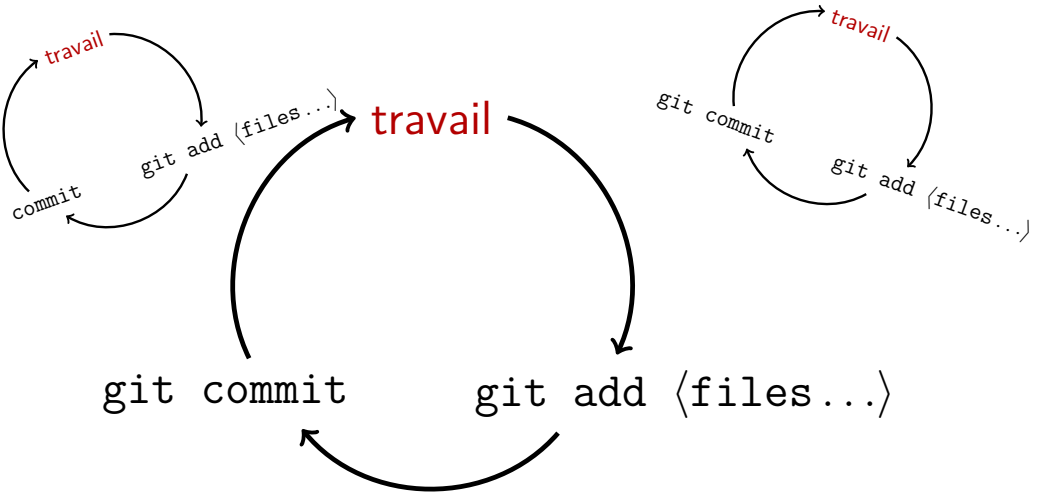
on parle de clone.



Système distribué

Chaque collaborateur a sa propre “copie” du dépôt git :

on parle de clone.

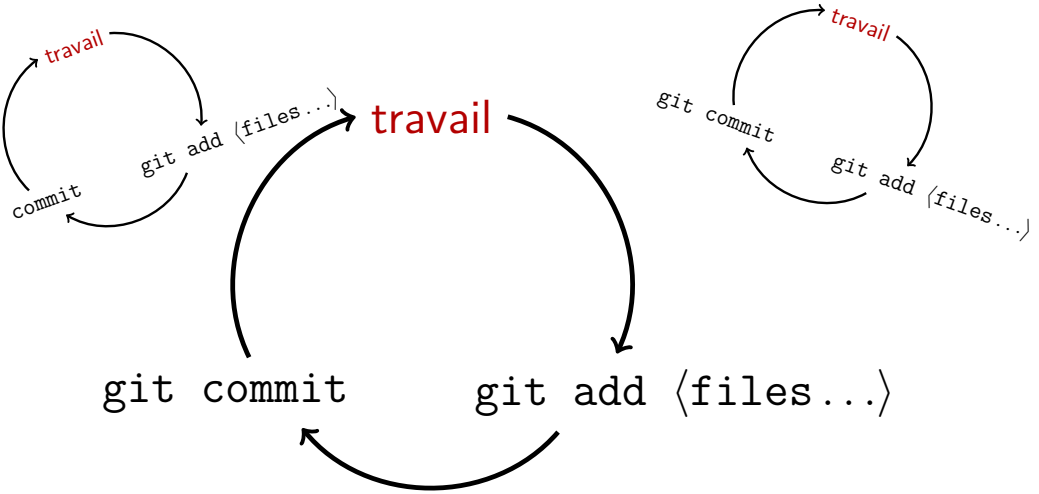


Le travail se fait localement, chaque collaborateur faisant évoluer son clone indépendamment des autres.

Système distribué

Chaque collaborateur a sa propre “copie” du dépôt git :

on parle de clone.



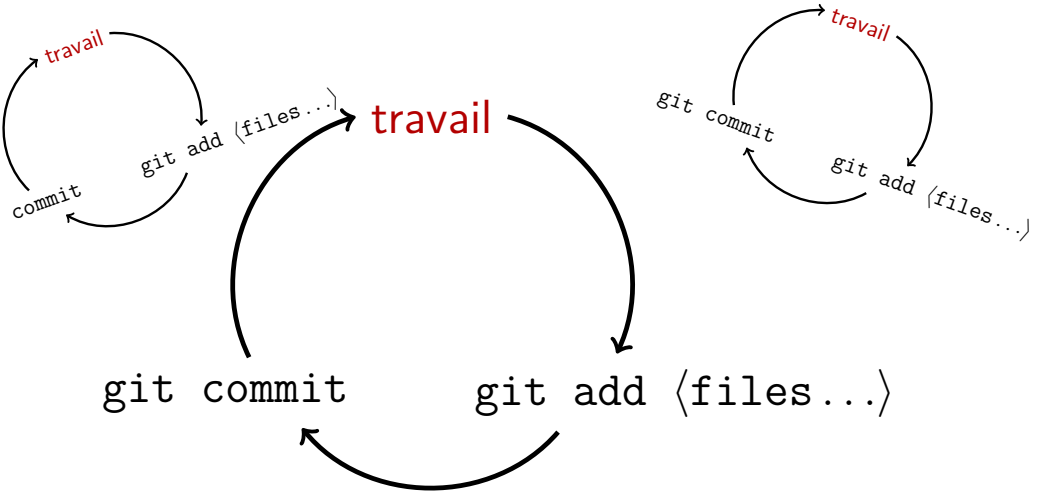
Le travail se fait localement, chaque collaborateur faisant évoluer son clone indépendamment des autres.

Comment collaborer alors ?

Système distribué

Chaque collaborateur a sa propre “copie” du dépôt git :

on parle de clone.



Le travail se fait localement, chaque collaborateur faisant évoluer son clone indépendamment des autres.

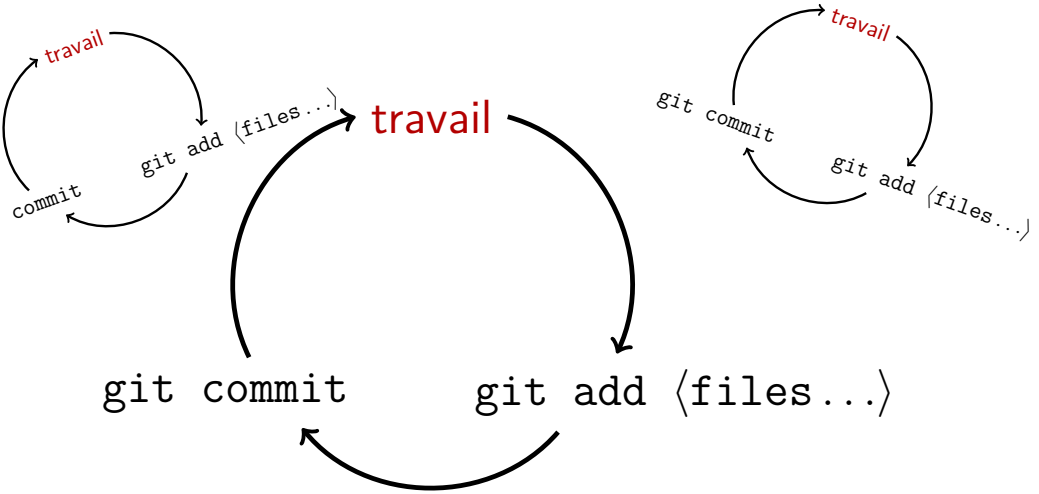
Comment collaborer alors ?

- Comment obtenir son propre clone ?
- Comment se synchroniser avec les autres ?

Système distribué

Chaque collaborateur a sa propre “copie” du dépôt git :

on parle de clone.



Le travail se fait localement, chaque collaborateur faisant évoluer son clone indépendamment des autres.

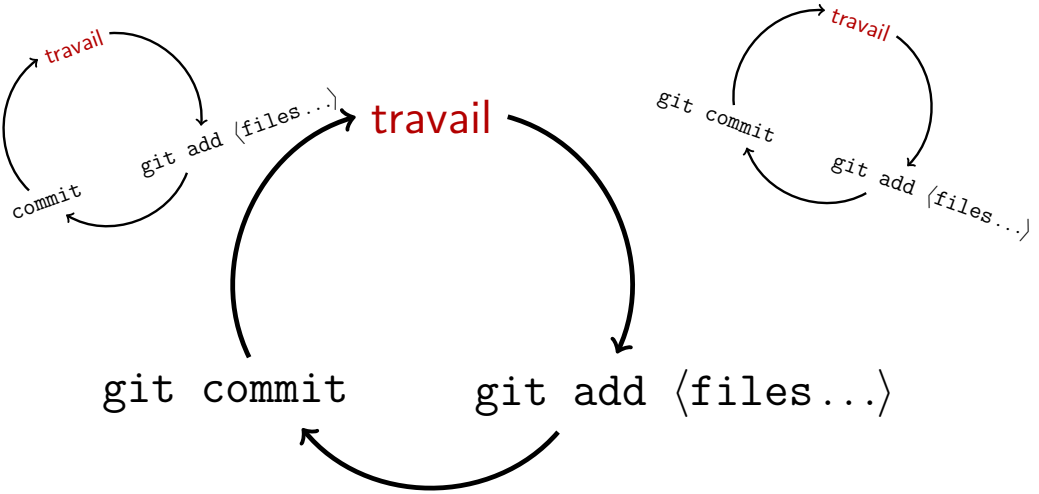
Comment collaborer alors ?

- Comment obtenir son propre clone ?
- Comment se synchroniser avec les autres ?
  - Comment récupérer le travail des autres ?
  - Comment envoyer son propre travail aux autres ?

Système distribué

Chaque collaborateur a sa propre “copie” du dépôt git :

on parle de clone.



Le travail se fait localement, chaque collaborateur faisant évoluer son clone indépendamment des autres.

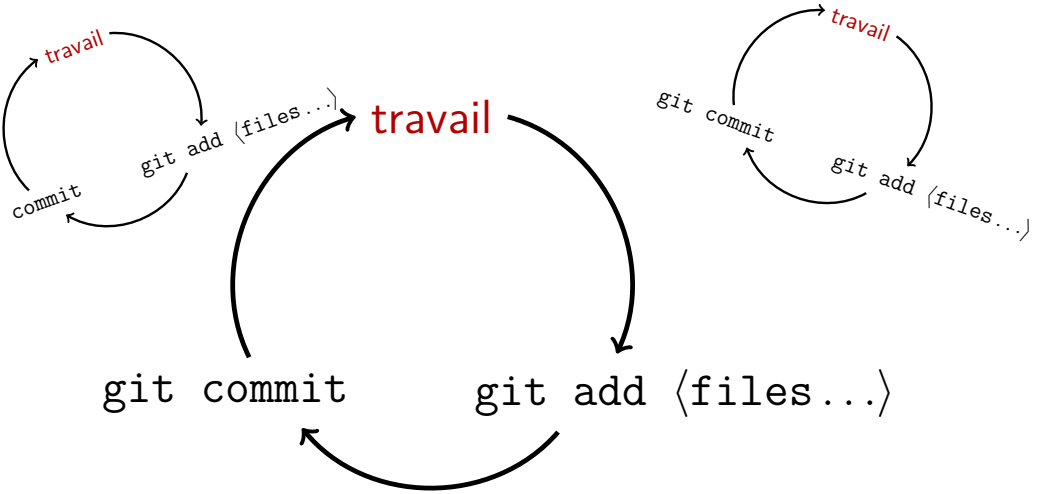
Comment collaborer alors ?

- Comment obtenir son propre clone ?
- Comment se synchroniser avec les autres ?
  - Comment récupérer le travail des autres ?
  - Comment envoyer son propre travail aux autres ?
- Quels problèmes peuvent survenir, et comment les résoudre ?

Système distribué

Chaque collaborateur a sa propre “copie” du dépôt git :

on parle de clone.



Le travail se fait localement, chaque collaborateur faisant évoluer son clone indépendamment des autres.

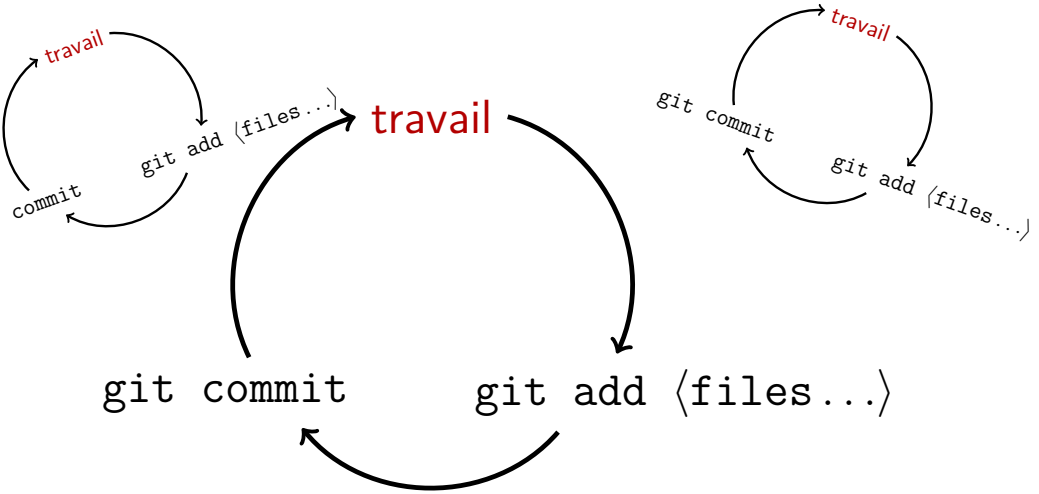
Comment collaborer alors ?

- Comment obtenir son propre clone ? → `git clone`
- Comment se synchroniser avec les autres ?
  - Comment récupérer le travail des autres ?
  - Comment envoyer son propre travail aux autres ?
- Quels problèmes peuvent survenir, et comment les résoudre ?

Système distribué

Chaque collaborateur a sa propre “copie” du dépôt git :

on parle de clone.



Le travail se fait localement, chaque collaborateur faisant évoluer son clone indépendamment des autres.

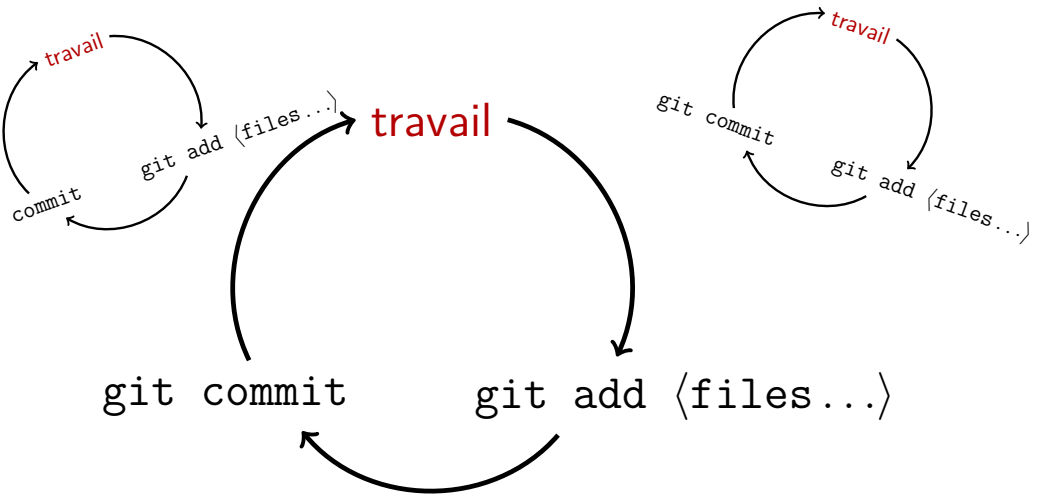
Comment collaborer alors ?

- Comment obtenir son propre clone ? → `git clone`
- Comment se synchroniser avec les autres ?
  - Comment récupérer le travail des autres ? → `git pull`
  - Comment envoyer son propre travail aux autres ?
- Quels problèmes peuvent survenir, et comment les résoudre ?

Système distribué

Chaque collaborateur a sa propre “copie” du dépôt git :

on parle de clone.



Le travail se fait localement, chaque collaborateur faisant évoluer son clone indépendamment des autres.

Comment collaborer alors ?

- Comment obtenir son propre clone ? → `git clone`
- Comment se synchroniser avec les autres ?
  - Comment récupérer le travail des autres ? → `git pull`
  - Comment envoyer son propre travail aux autres ? → `git push`
- Quels problèmes peuvent survenir, et comment les résoudre ?

Obtenir son clone

La commande `git clone` permet d'obtenir un clone d'un dépôt.

Obtenir son clone

La commande `git clone` permet d'obtenir un clone d'un dépôt.

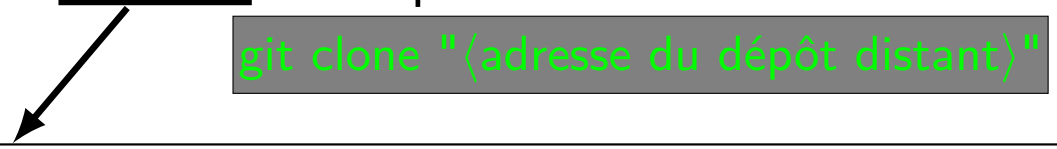
Elle attend un argument : l'adresse du dépôt à cloner :

```
git clone "⟨adresse du dépôt distant⟩"
```

Obtenir son clone

La commande `git clone` permet d'obtenir un clone d'un dépôt.

Elle attend un argument : l'adresse du dépôt à cloner :

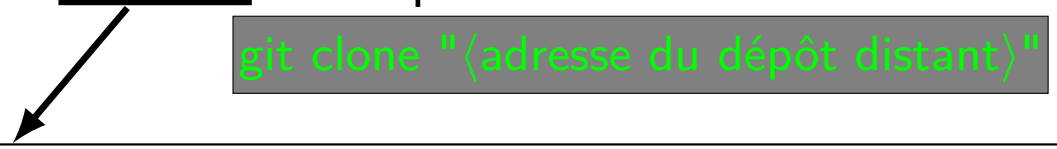


- chemin vers un dépôt sur notre machine (*e.g.*, `~/original/`);
- chemin vers un dépôt sur une machine distante (*e.g.*, *via* `ssh`);
- `url` vers un dépôt sur un serveur dédié (*e.g.*, gitlab).

Obtenir son clone

La commande `git clone` permet d'obtenir un clone d'un dépôt.

Elle attend un argument : l'adresse du dépôt à cloner :



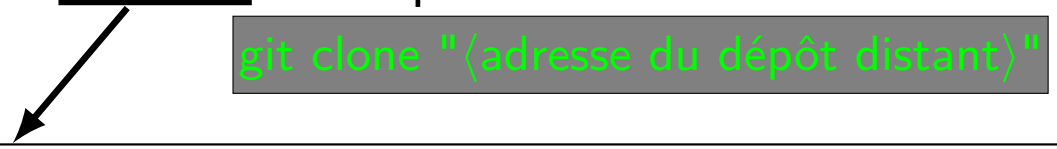
- chemin vers un dépôt sur notre machine (*e.g.*, `~/original/`);
- chemin vers un dépôt sur une machine distante (*e.g.*, *via* `ssh`);
- `url` vers un dépôt sur un serveur dédié (*e.g.*, gitlab).

On utilisera surtout gitlab.

Obtenir son clone

La commande `git clone` permet d'obtenir un clone d'un dépôt.

Elle attend un argument : l'adresse du dépôt à cloner :



- chemin vers un dépôt sur notre machine (*e.g.*, `~/original/`);
- chemin vers un dépôt sur une machine distante (*e.g.*, *via* `ssh`);
- `url` vers un dépôt sur un serveur dédié (*e.g.*, gitlab).

On utilisera surtout gitlab.

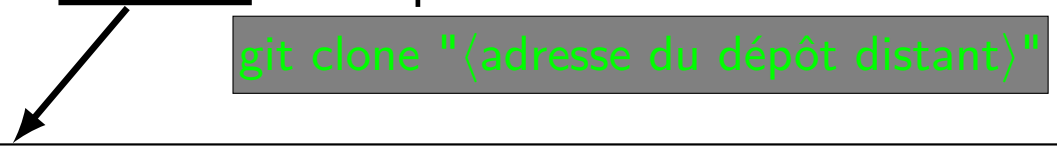
Dépôt gitlab (ou autre) :

- soit partagé par quelqu'un ;
- soit créé par nous même.

Obtenir son clone

La commande `git clone` permet d'obtenir un clone d'un dépôt.

Elle attend un argument : l'adresse du dépôt à cloner :



- chemin vers un dépôt sur notre machine (*e.g.*, `~/original/`);
- chemin vers un dépôt sur une machine distante (*e.g.*, *via* `ssh`);
- `url` vers un dépôt sur un serveur dédié (*e.g.*, gitlab).

On utilisera surtout gitlab.

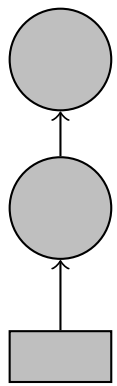
Dépôt gitlab (ou autre) :

- soit partagé par quelqu'un ;
- soit créé par nous même.

Créer son dépôt sur gitlab

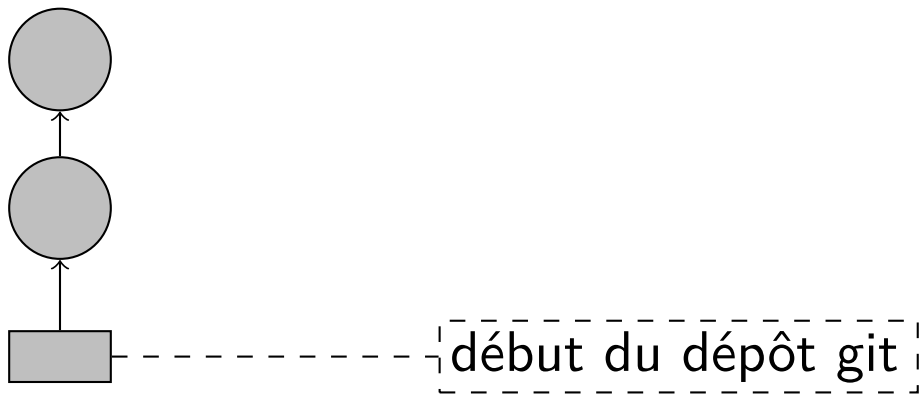
Il faut créer un projet sur gitlab, *c.f.* TP précédent et prochains.

git push



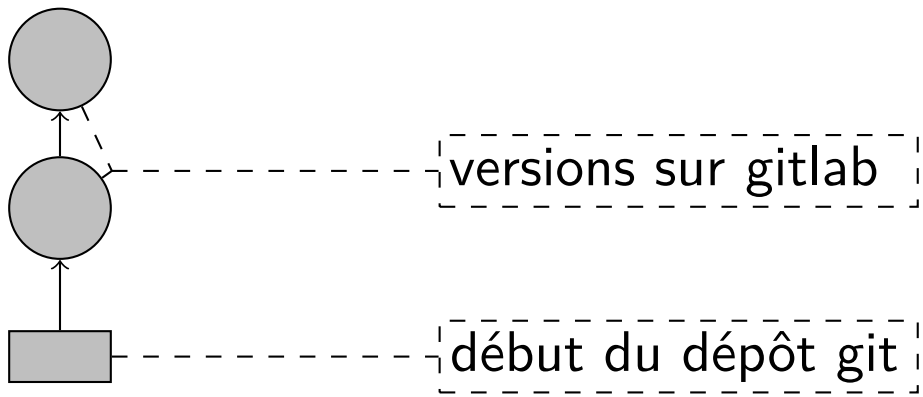
: versions sur le dépôt distant

git push



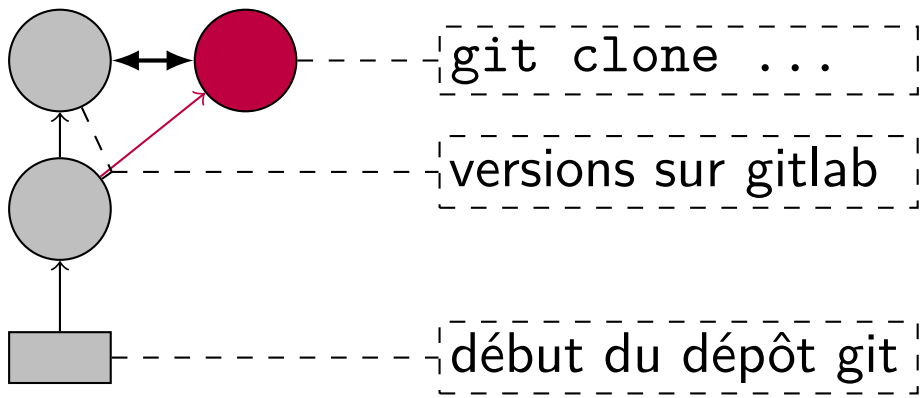
: versions sur le dépôt distant

git push



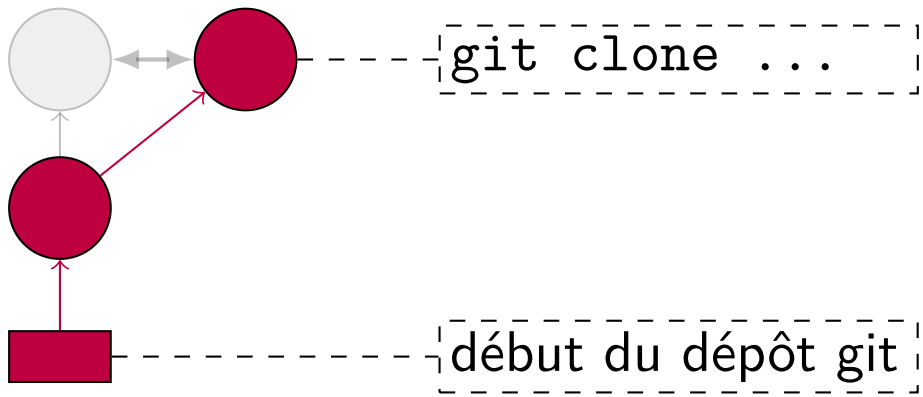
: versions sur le dépôt distant

git push



: versions sur le dépôt distant – : versions sur le dépôt local

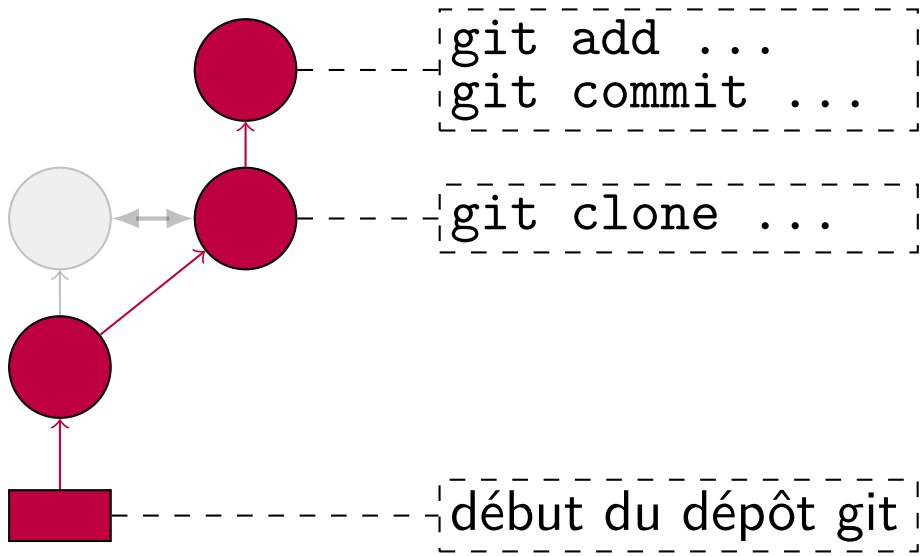
git push

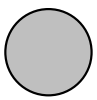
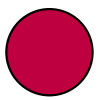


: versions sur le dépôt distant – : versions sur le dépôt local

Ce qu'on voit sur le dépôt local.

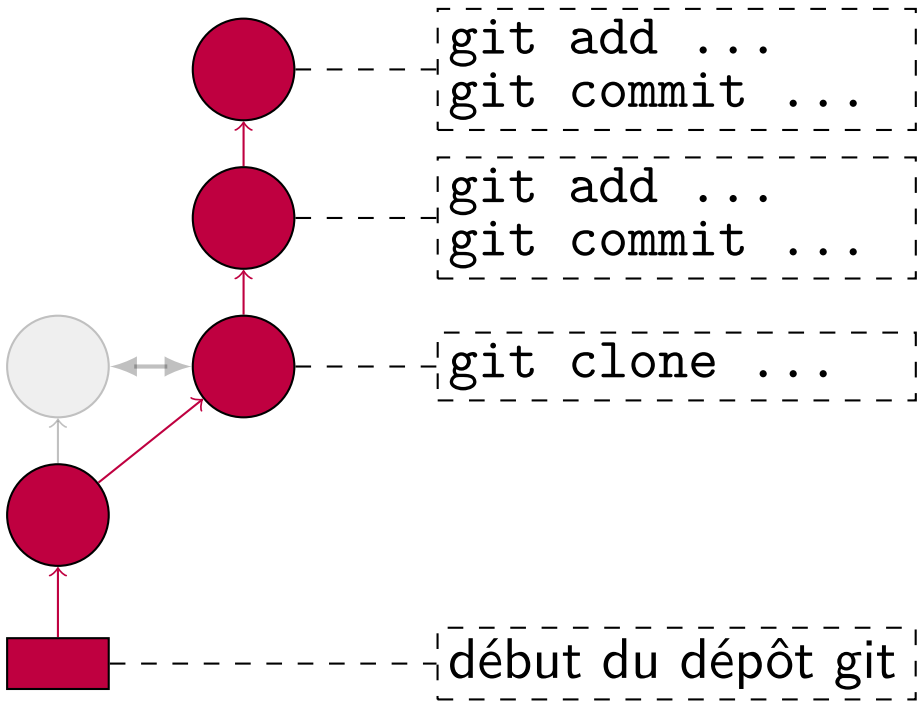
git push



 : versions sur le dépôt distant –  : versions sur le dépôt local

Ce qu'on voit sur le dépôt local.

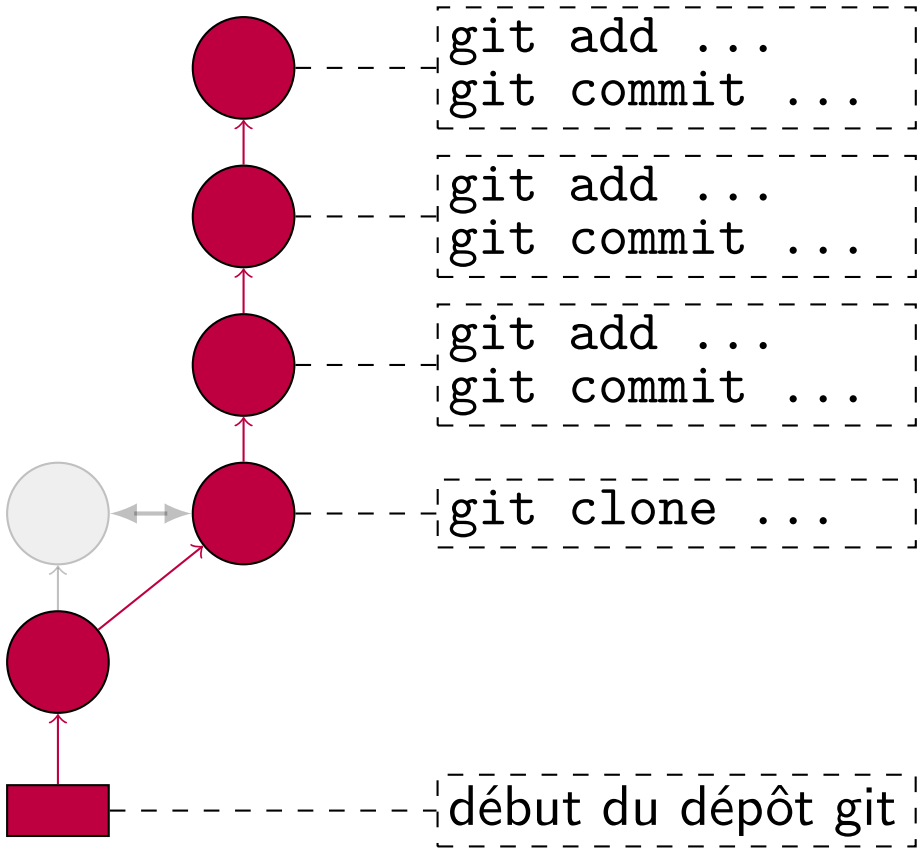
git push



: versions sur le dépôt distant – : versions sur le dépôt local

Ce qu'on voit sur le dépôt local.

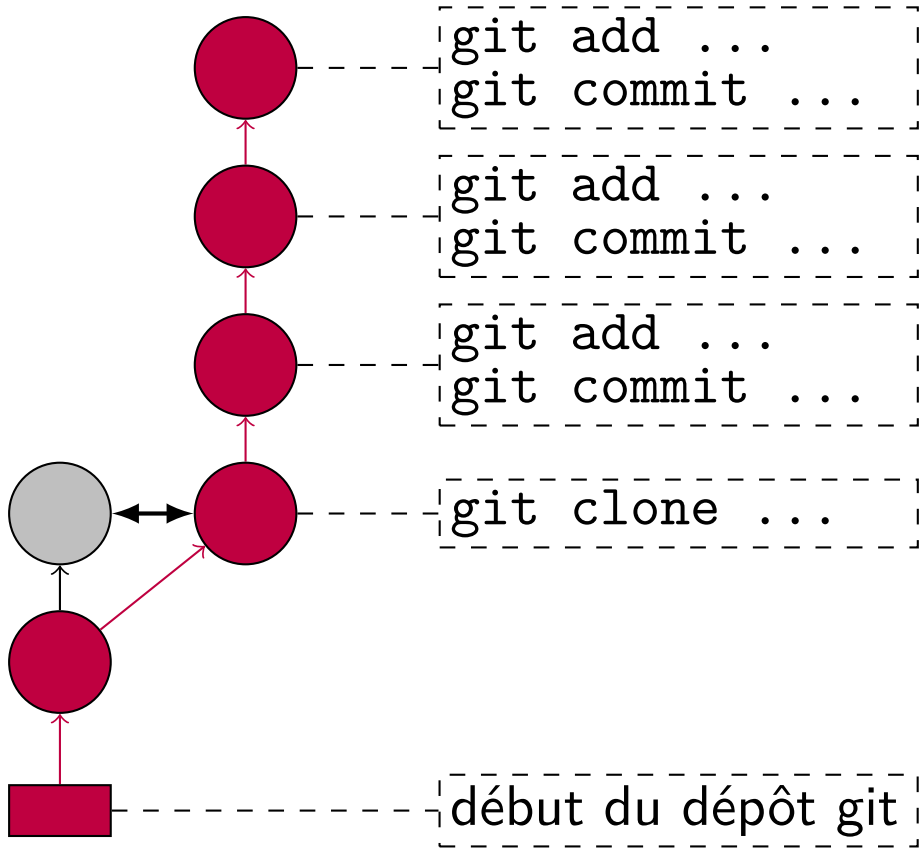
git push



: versions sur le dépôt distant – : versions sur le dépôt local

Ce qu'on voit sur le dépôt local.

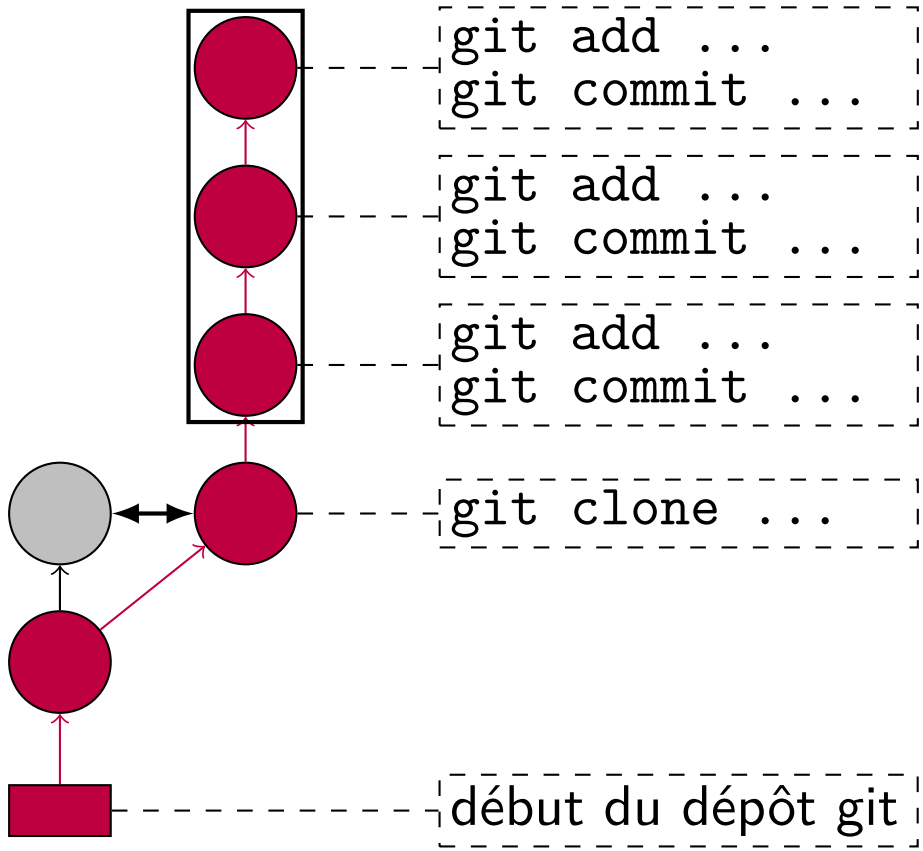
git push



: versions sur le dépôt distant – : versions sur le dépôt local

Le dépôt distant est resté en arrière...

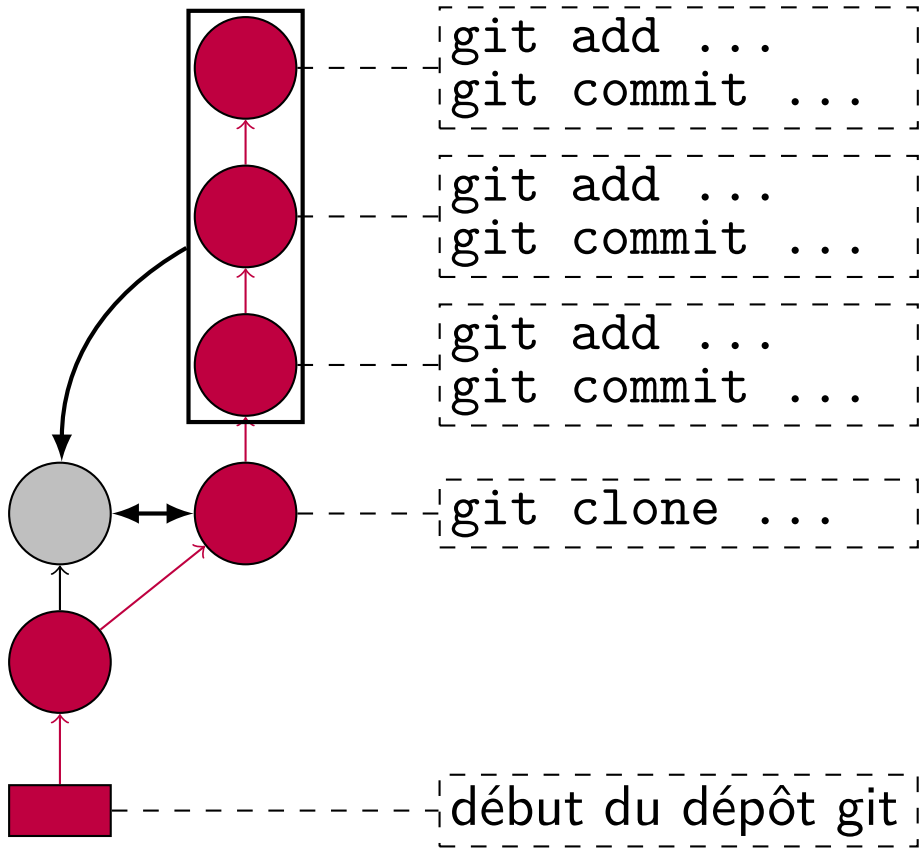
git push



: versions sur le dépôt distant – : versions sur le dépôt local

Le dépôt distant est resté en arrière...

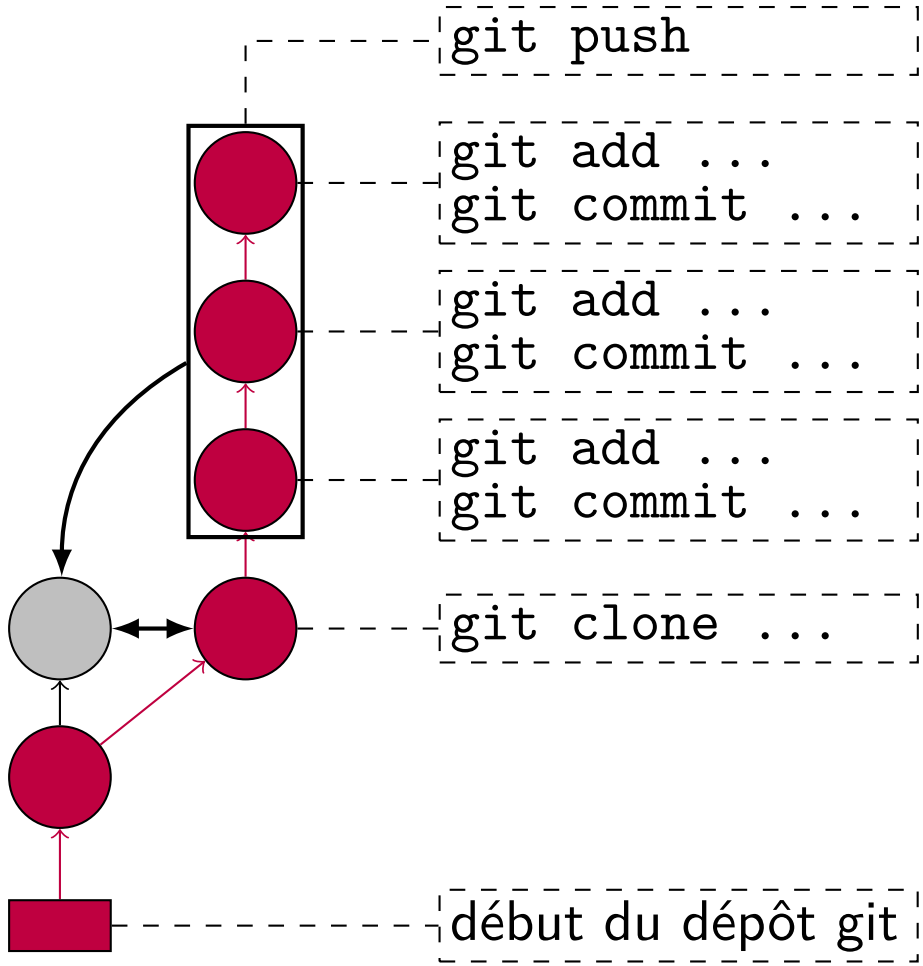
git push



: versions sur le dépôt distant – : versions sur le dépôt local

Le dépôt distant est resté en arrière...

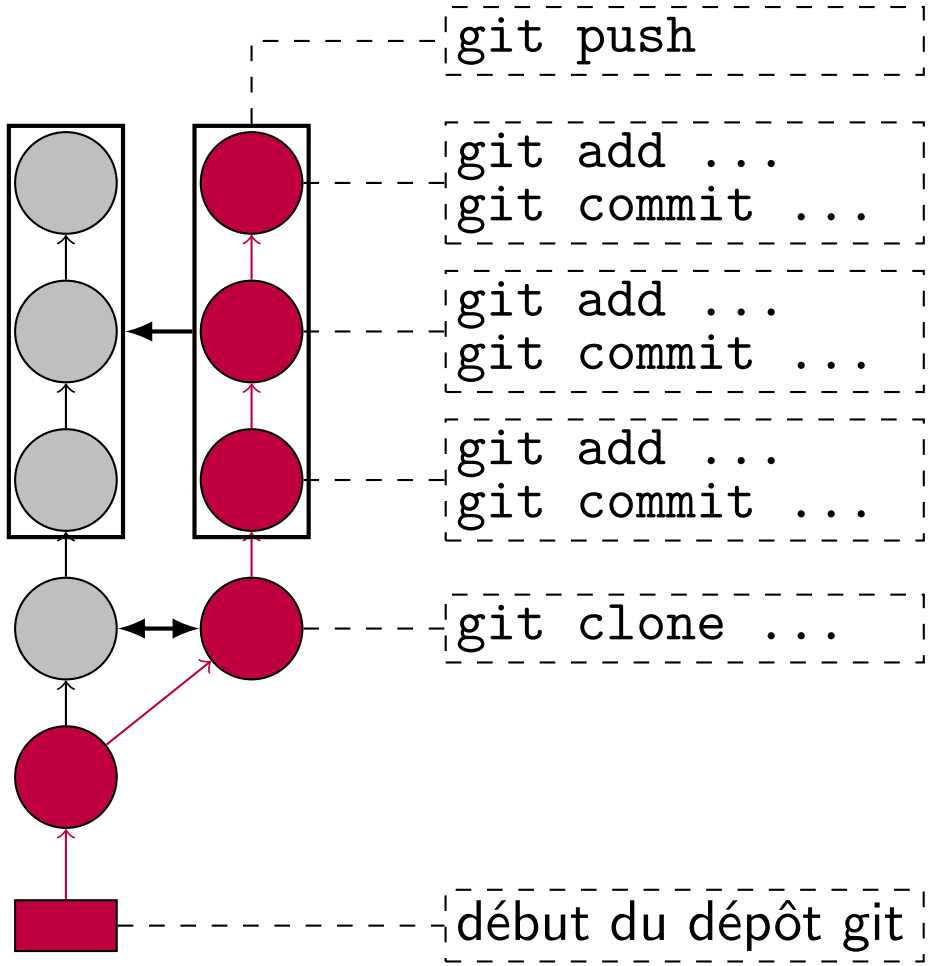
git push



: versions sur le dépôt distant – : versions sur le dépôt local

On “*pousse*” nos versions sur le dépôt distants.

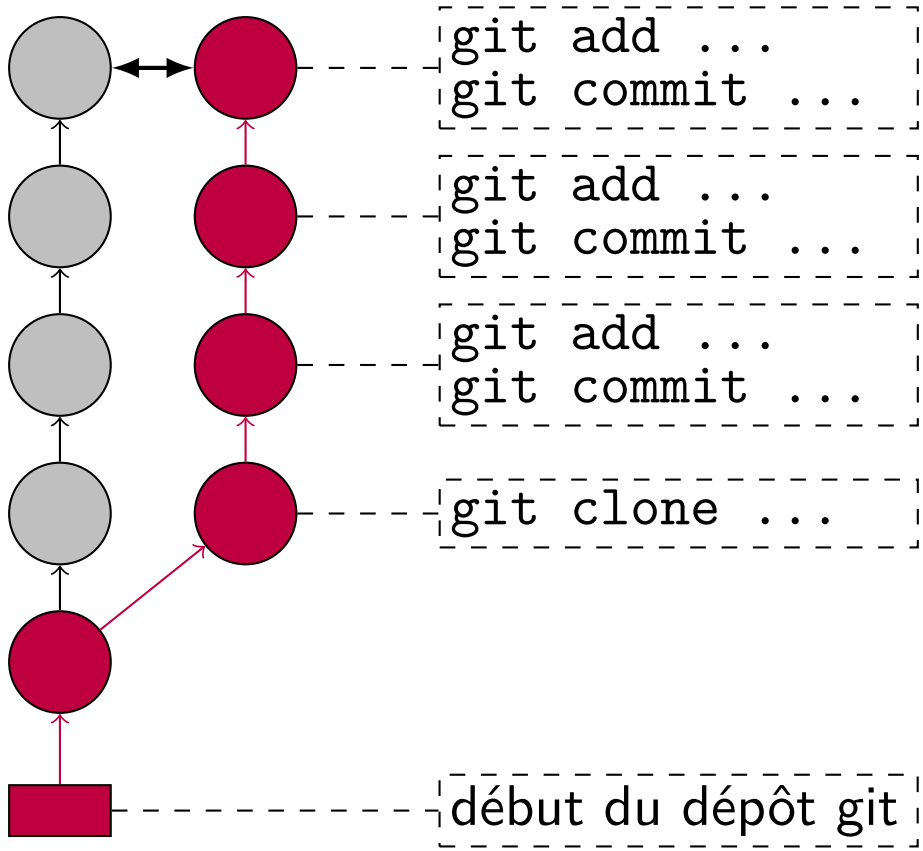
git push



: versions sur le dépôt distant – : versions sur le dépôt local

On “*pousse*” nos versions sur le dépôt distants.

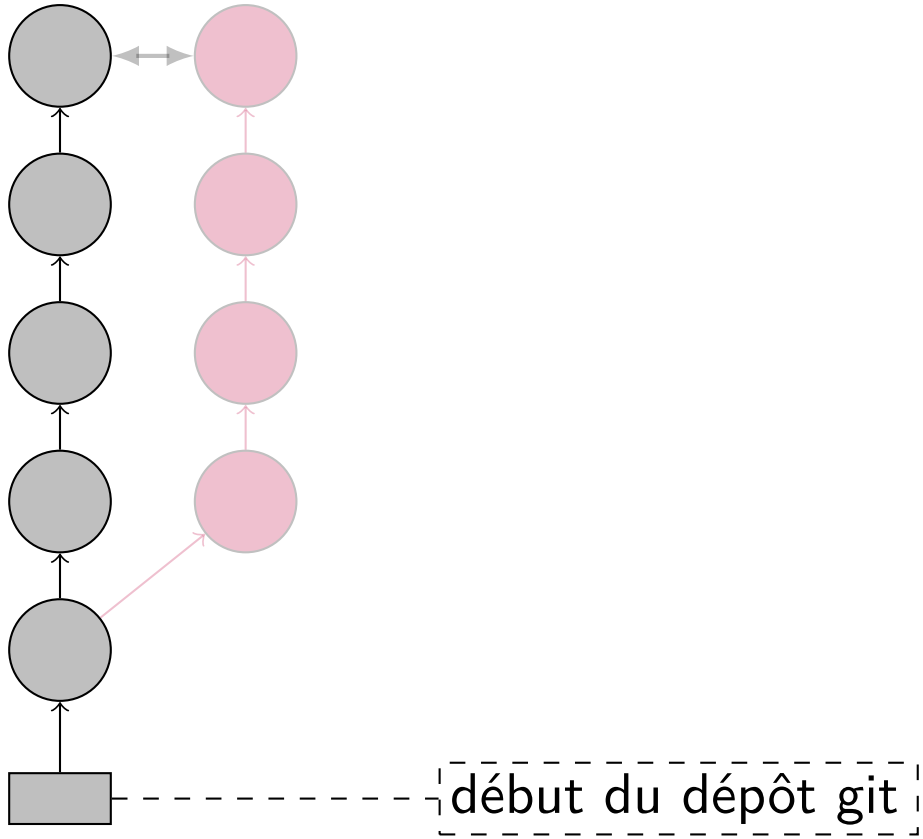
git push



: versions sur le dépôt distant – : versions sur le dépôt local

On “*pousse*” nos versions sur le dépôt distants.

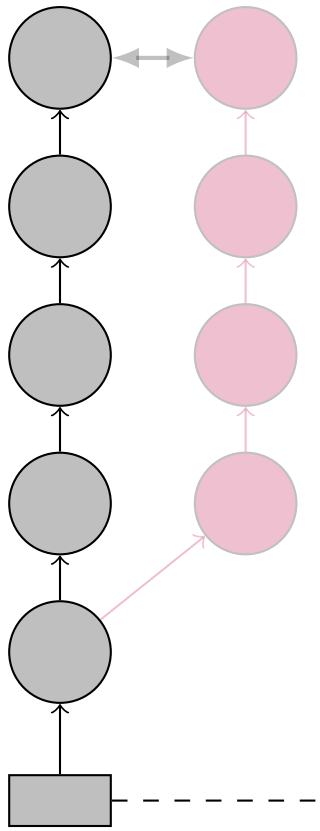
git push



● : versions sur le dépôt distant – ● : versions sur le dépôt local

Ce qu'on voit sur le dépôt distant.

`git push`



`git push` nécessite :

- dépôt distant configuré localement

  c'est automatique lors du `git clone` ;
  aide sur gitlab sinon ;

- droits d'écriture sur le dépôt distant :

  admin. du projet peut les octroyer ;
  c.f. prochains TP et projet.

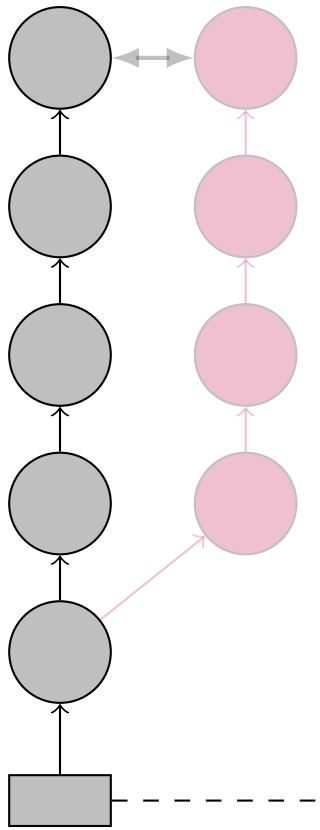
- la dernière version sur le dépôt distant

  doit être celle de laquelle on est parti.
  c.f. slide 13 à venir. . .

● : versions sur le dépôt distant – ● : versions sur le dépôt local

Ce qu'on voit sur le dépôt distant.

`git push`



`git push` nécessite :

- dépôt distant configuré localement

  c'est automatique lors du `git clone` ;
  aide sur gitlab sinon ;

- droits d'écriture sur le dépôt distant :

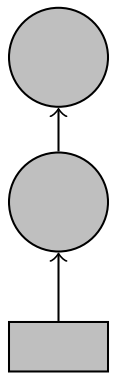
  admin. du projet peut les octroyer ;
  c.f. prochains TP et projet.

- la dernière version sur le dépôt distant doit être celle de laquelle on est parti.
  c.f. slide 13 à venir. . .

● : versions sur le dépôt distant – ● : versions sur le dépôt local

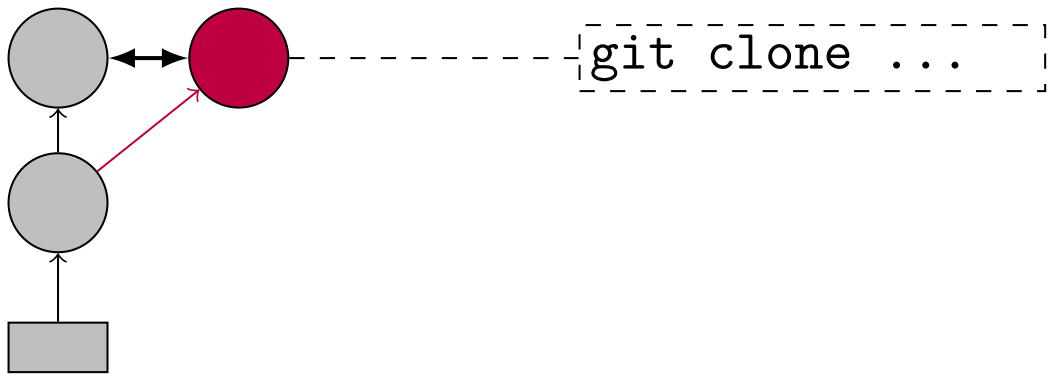
Ce qu'on voit sur le dépôt distant.

git pull



versions : ● sur dépôt distant

git pull

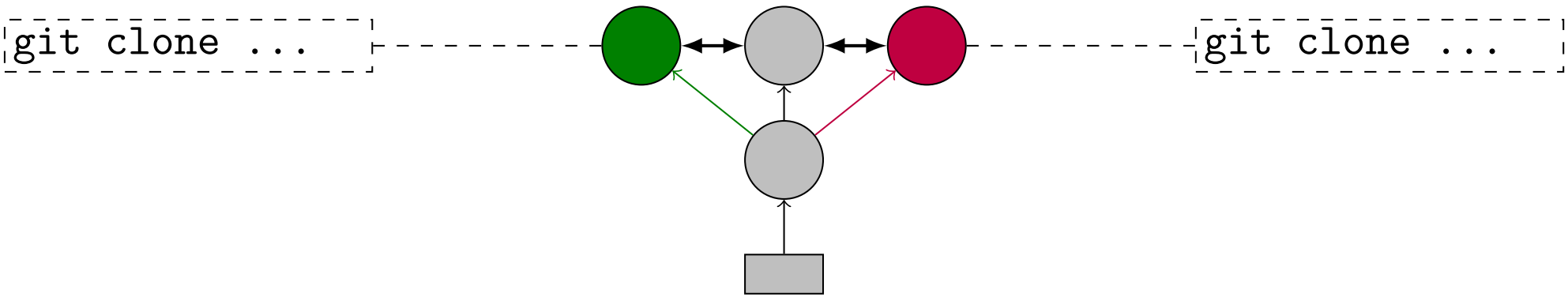


---

versions : sur dépôt distant – sur dépôt local

On clone le dépôt, mais on ne travaille pas tout de suite.

git pull

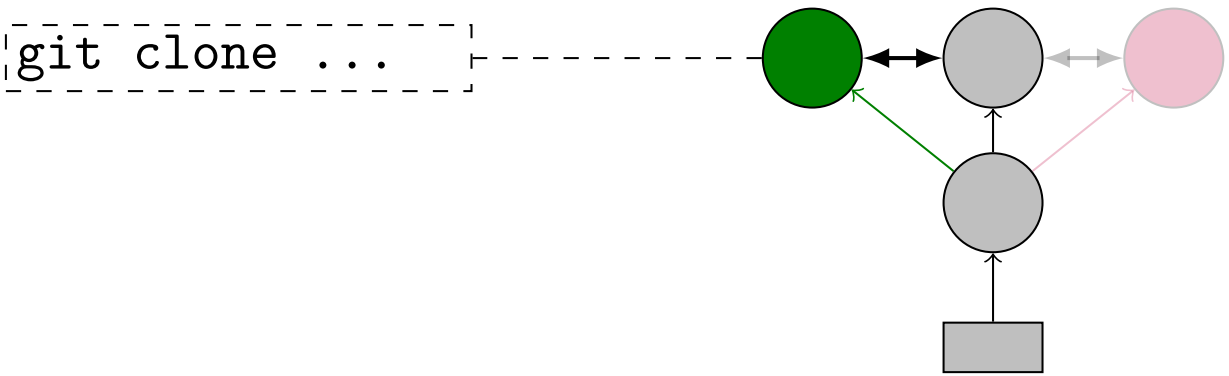


---

versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Un collaborateur décide de travailler sur le projet.

git pull

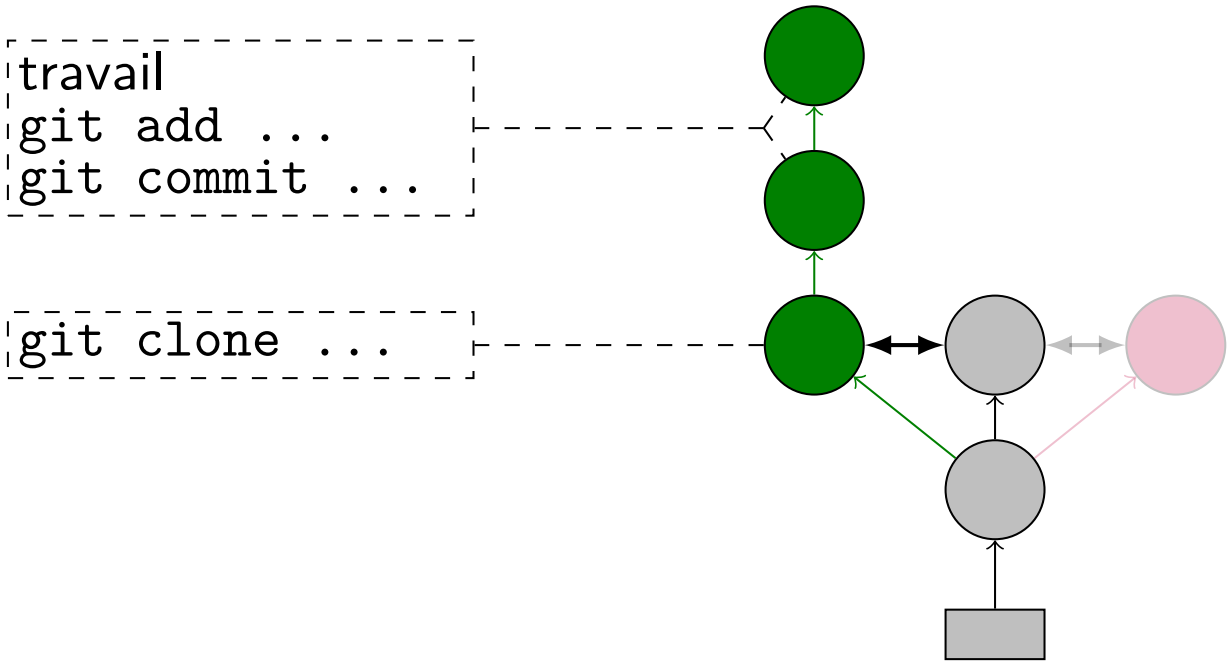


---

versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Un collaborateur décide de travailler sur le projet.

git pull

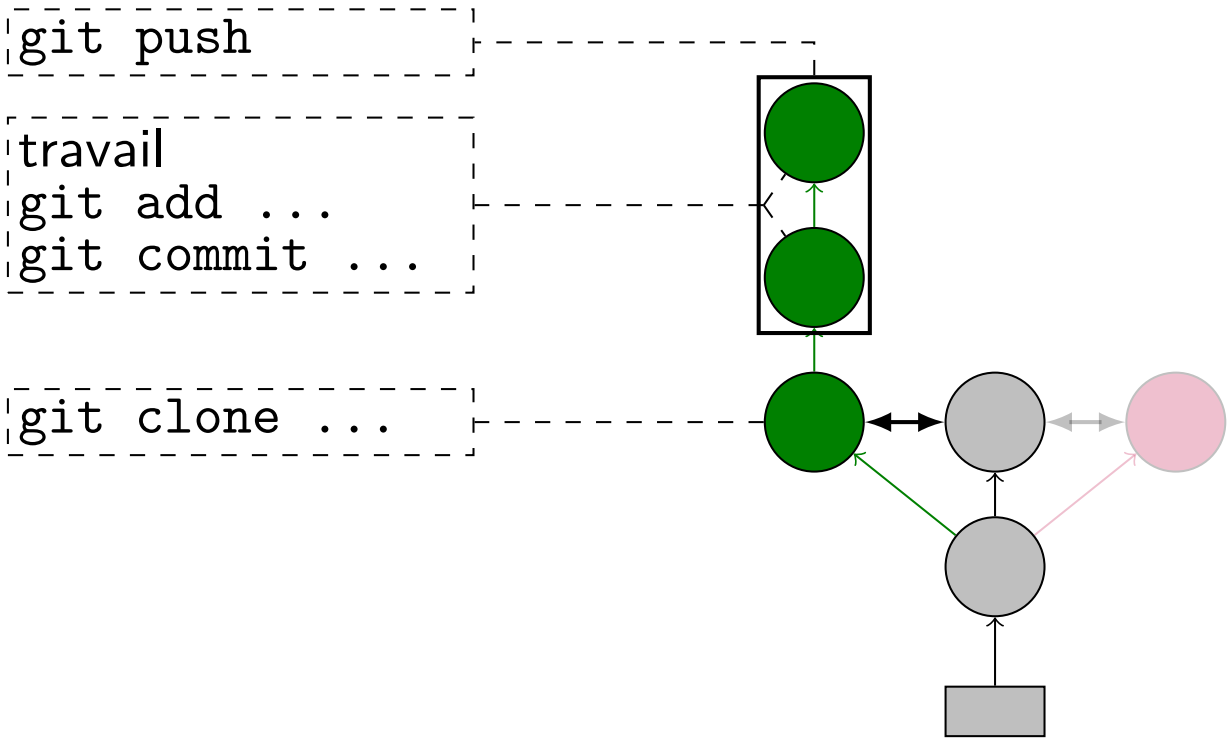


---

versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Un collaborateur décide de travailler sur le projet.

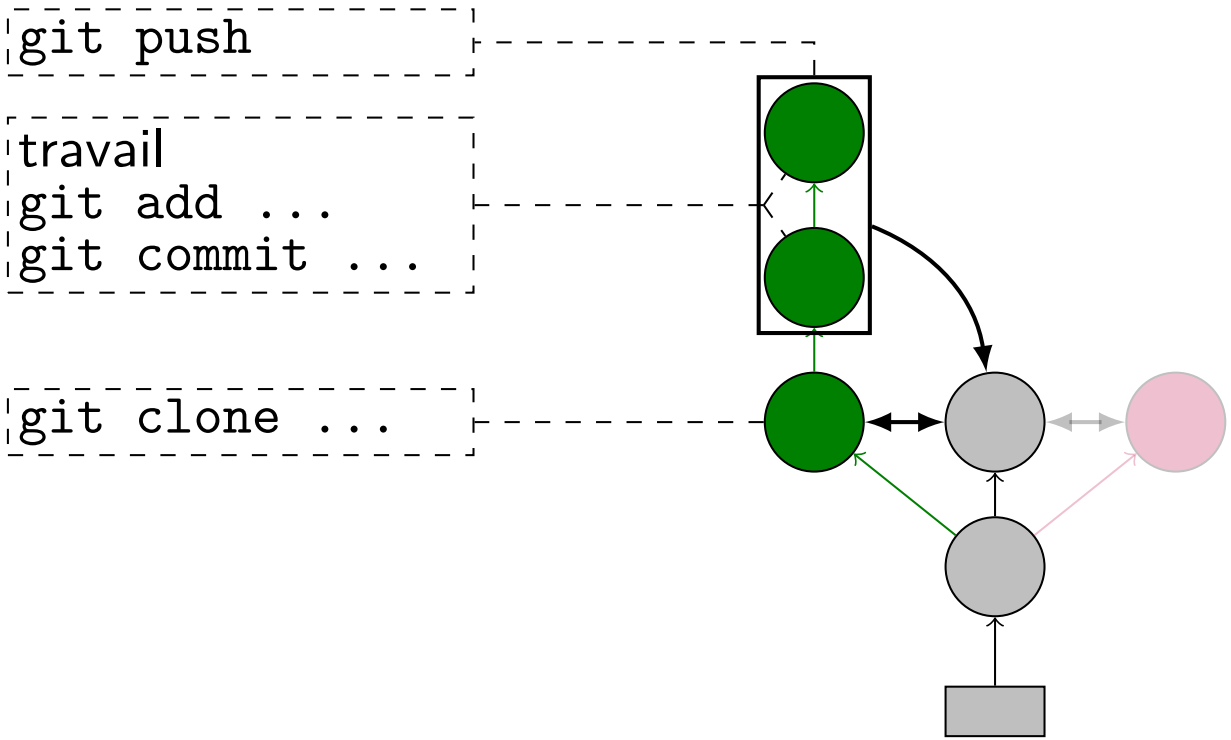
git pull



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Un collaborateur décide de travailler sur le projet.
Après quelques heures de travail, il pousse ses versions sur le dépôt distant.

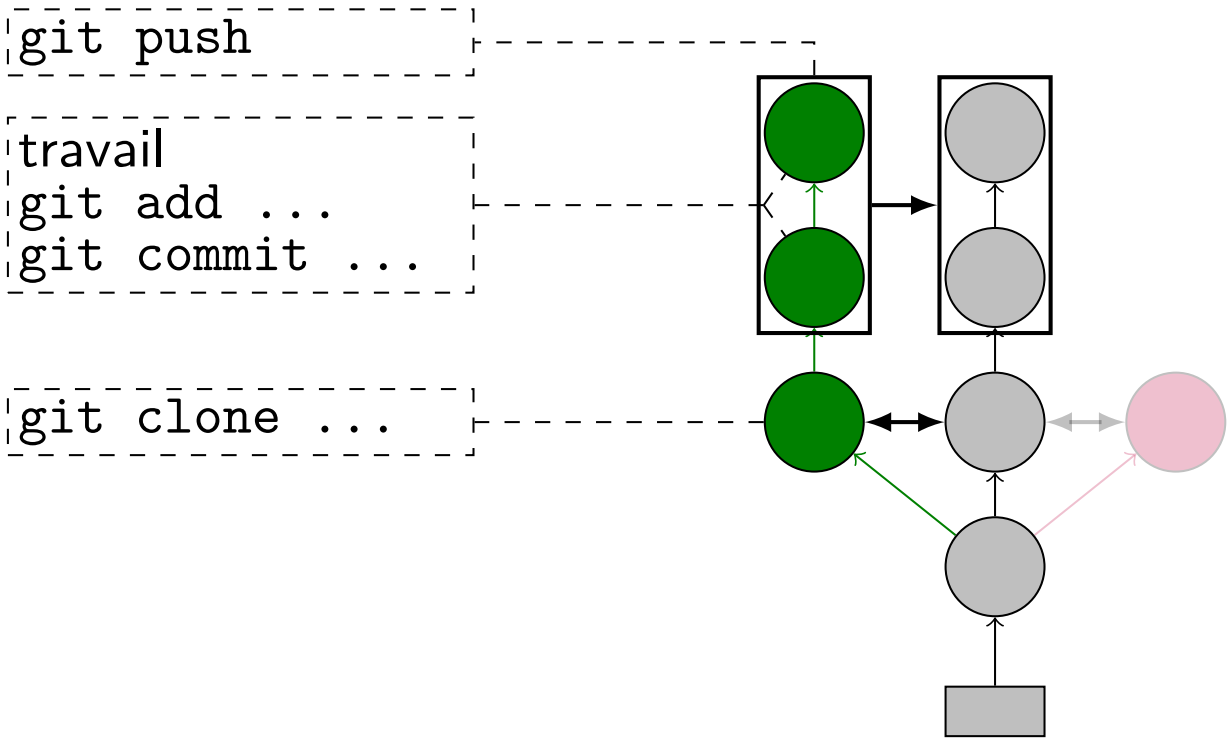
git pull



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Un collaborateur décide de travailler sur le projet.
Après quelques heures de travail, il pousse ses versions sur le dépôt distant.

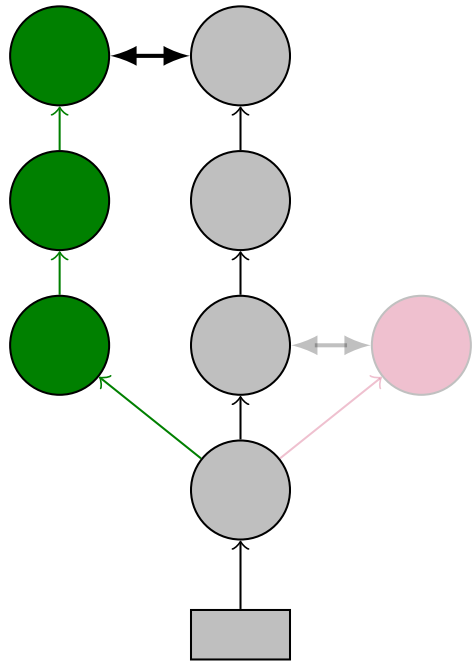
git pull



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Un collaborateur décide de travailler sur le projet.
Après quelques heures de travail, il pousse ses versions sur le dépôt distant.

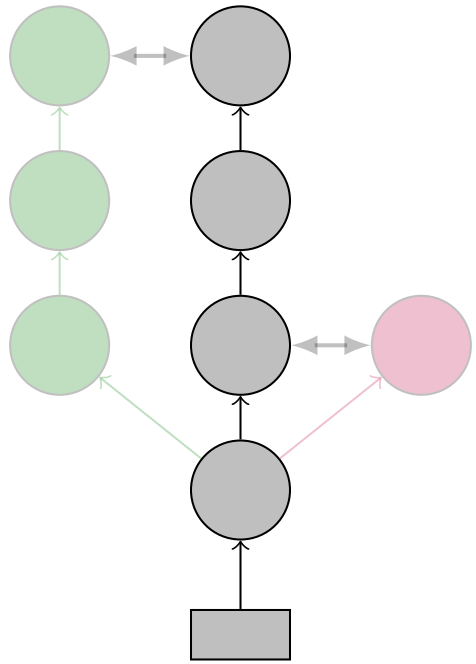
git pull



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

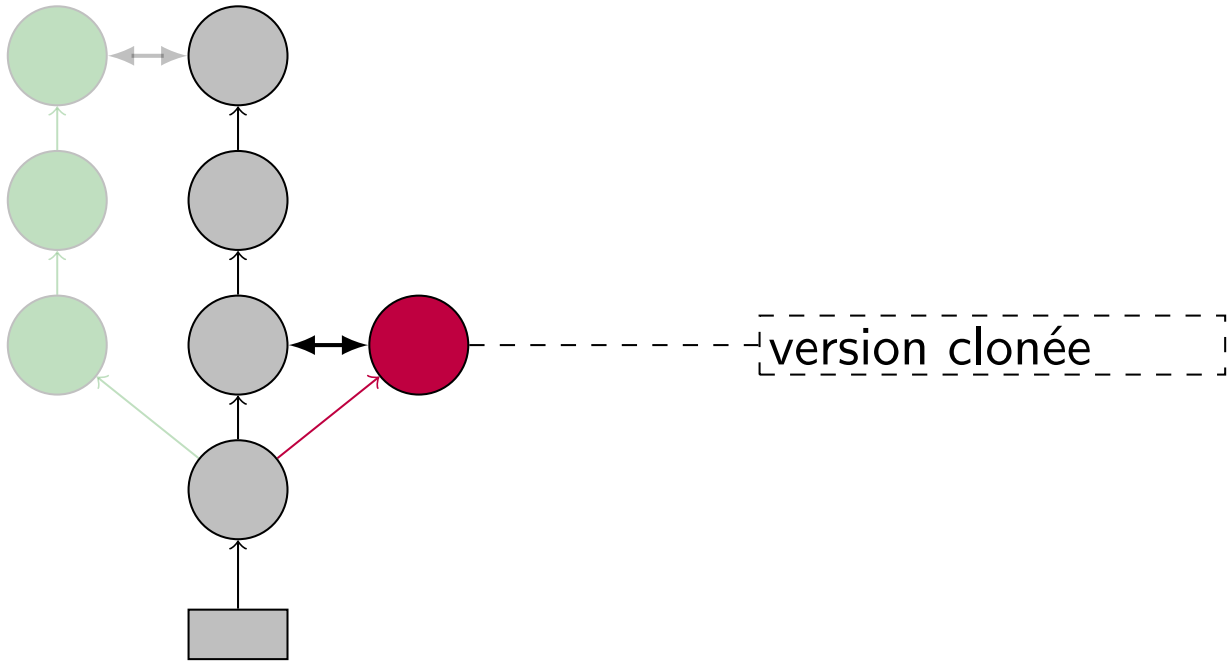
Un collaborateur décide de travailler sur le projet.
Après quelques heures de travail, il pousse ses versions sur le dépôt distant.

git pull



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

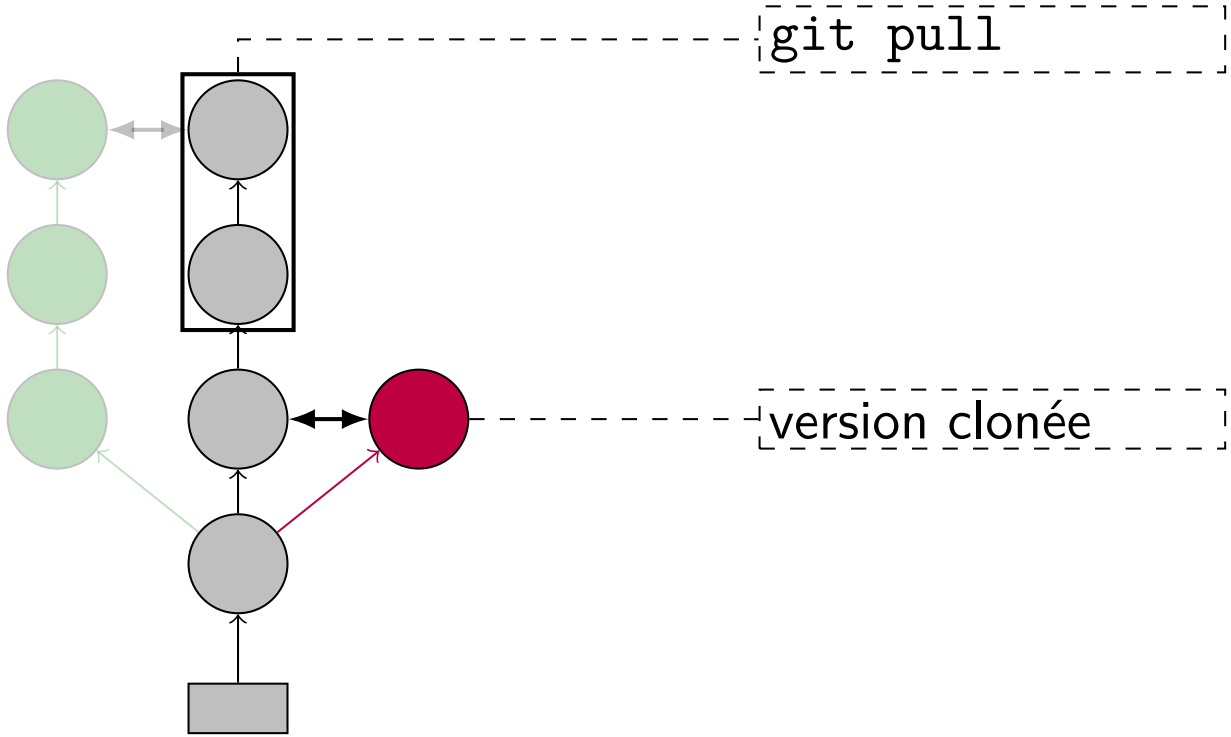
git pull



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Notre version, clonée plus tôt, n'est plus à jour.

git pull

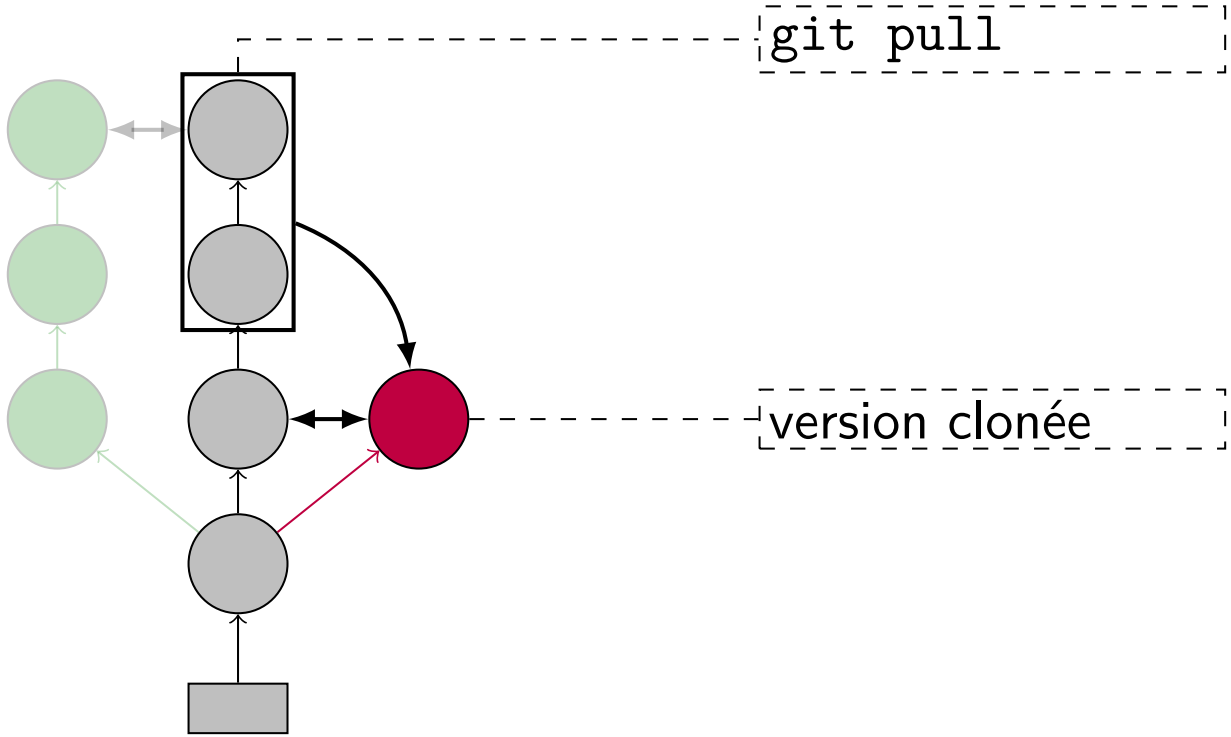


---

versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Notre version, clonée plus tôt, n'est plus à jour.
Pas besoin de refaire un clone, il suffit de faire `git pull`.

git pull

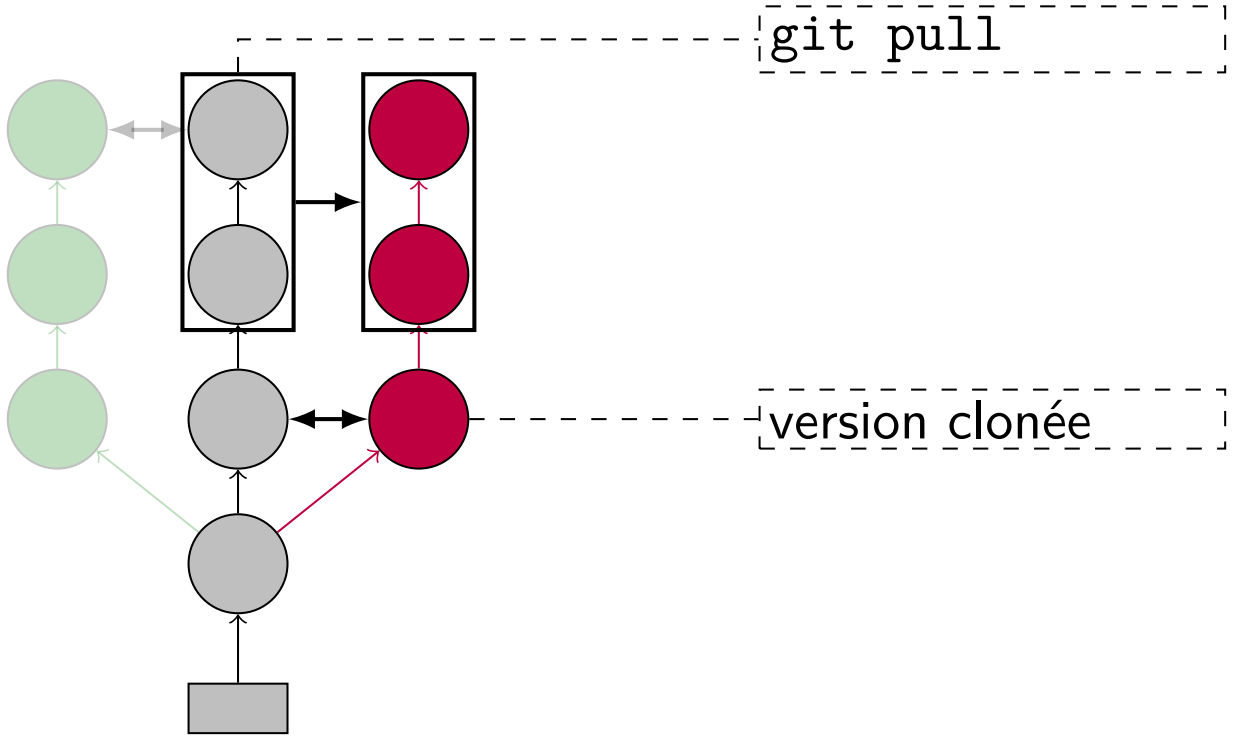


---

versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Notre version, clonée plus tôt, n'est plus à jour.
Pas besoin de refaire un clone, il suffit de faire `git pull`.

`git pull`

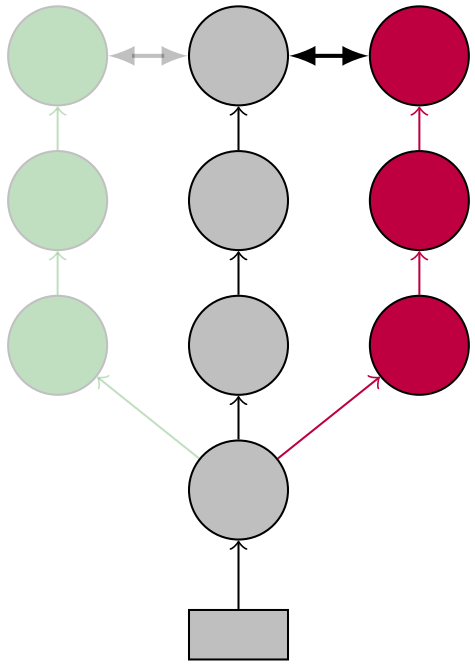


---

versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Notre version, clonée plus tôt, n'est plus à jour.
Pas besoin de refaire un clone, il suffit de faire `git pull`.

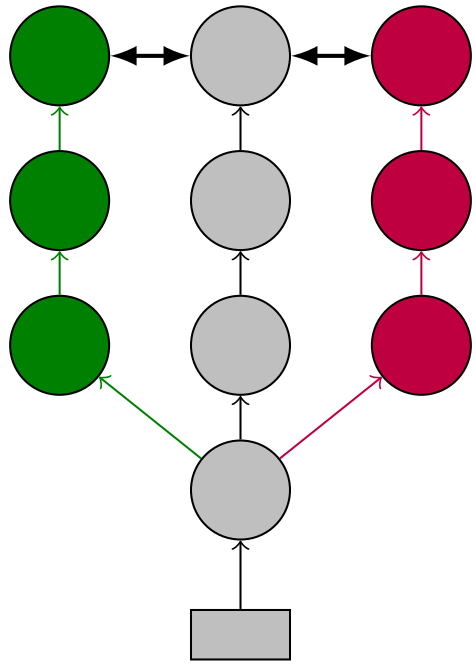
git pull



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Notre version, clonée plus tôt, n'est plus à jour.
Pas besoin de refaire un clone, il suffit de faire `git pull`.

`git pull`



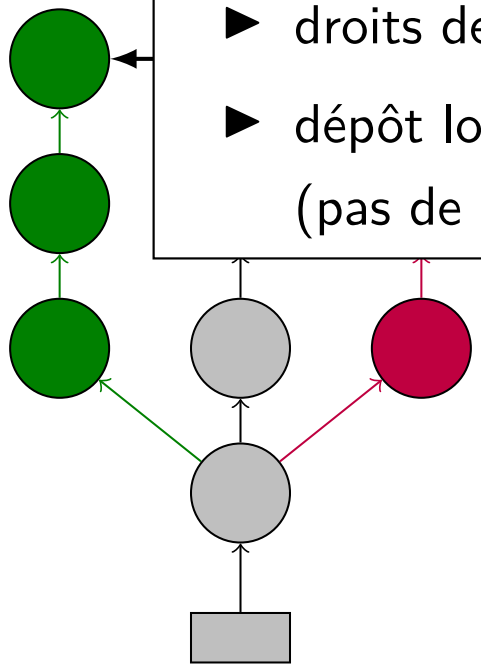
versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Notre version, clonée plus tôt, n'est plus à jour.
Pas besoin de refaire un clone, il suffit de faire `git pull`.

`git pull`

`git pull` nécessite :

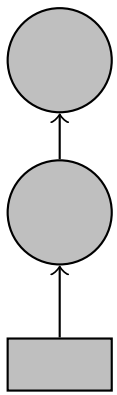
- dépôt distant configuré localement ;
- droits de lecture sur le dépôt distant ;
- dépôt local propre :
  (pas de modifications non-versionnées).



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

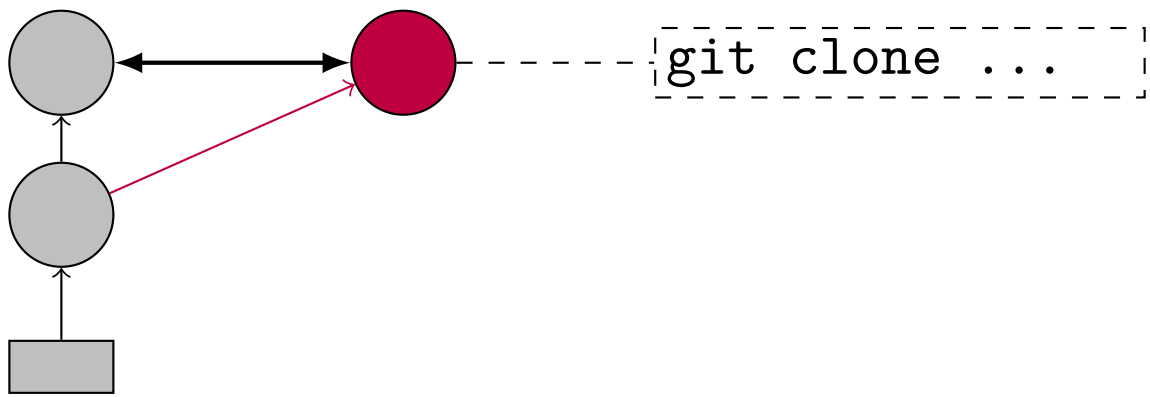
Notre version, clonée plus tôt, n'est plus à jour.
Pas besoin de refaire un clone, il suffit de faire `git pull`.

`git pull` avec fusion



versions : ◯ sur dépôt distant

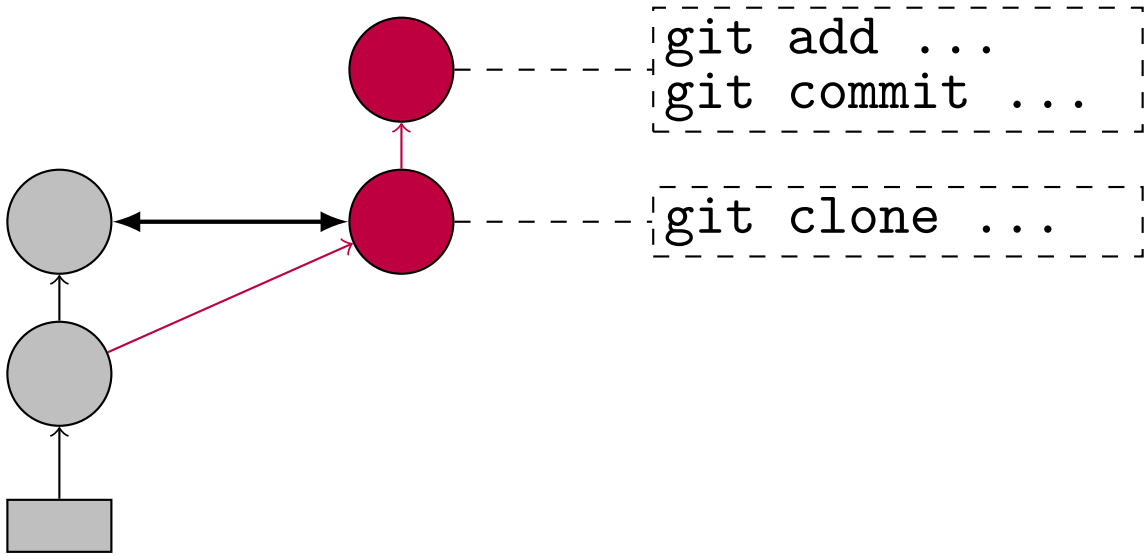
`git pull` avec fusion



versions : sur dépôt distant – sur dépôt local

On clone le dépôt, et on travaille sur le projet.

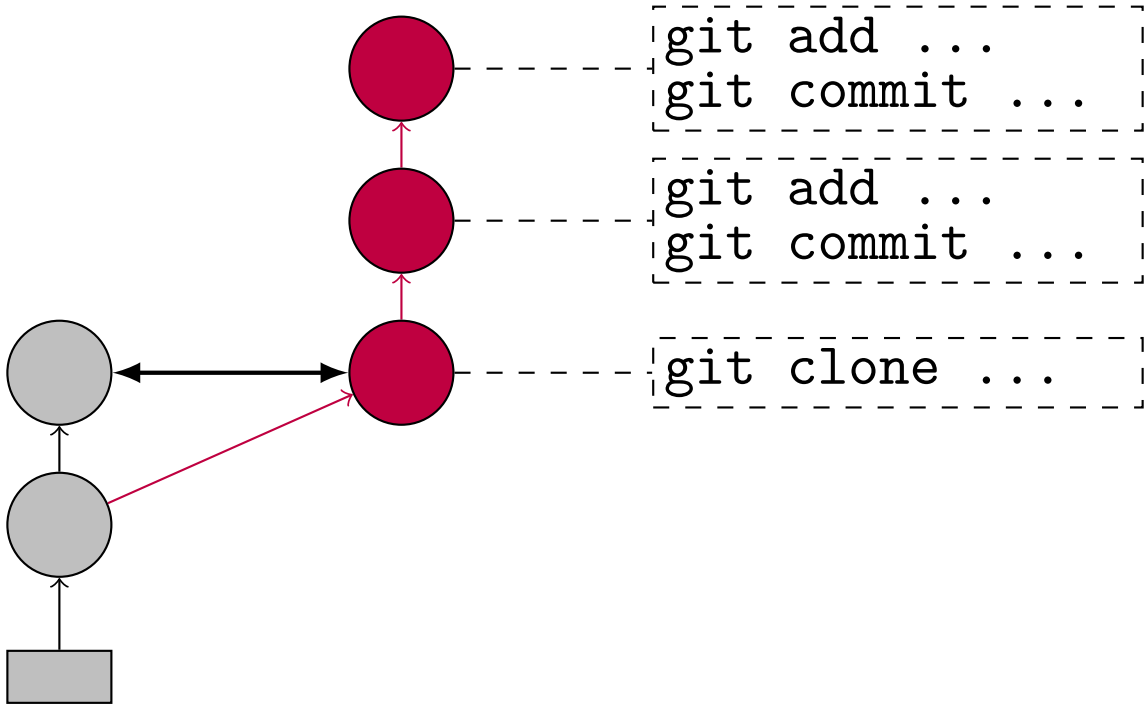
`git pull` avec fusion



versions : sur dépôt distant – sur dépôt local

On clone le dépôt, et on travaille sur le projet.

`git pull` avec fusion

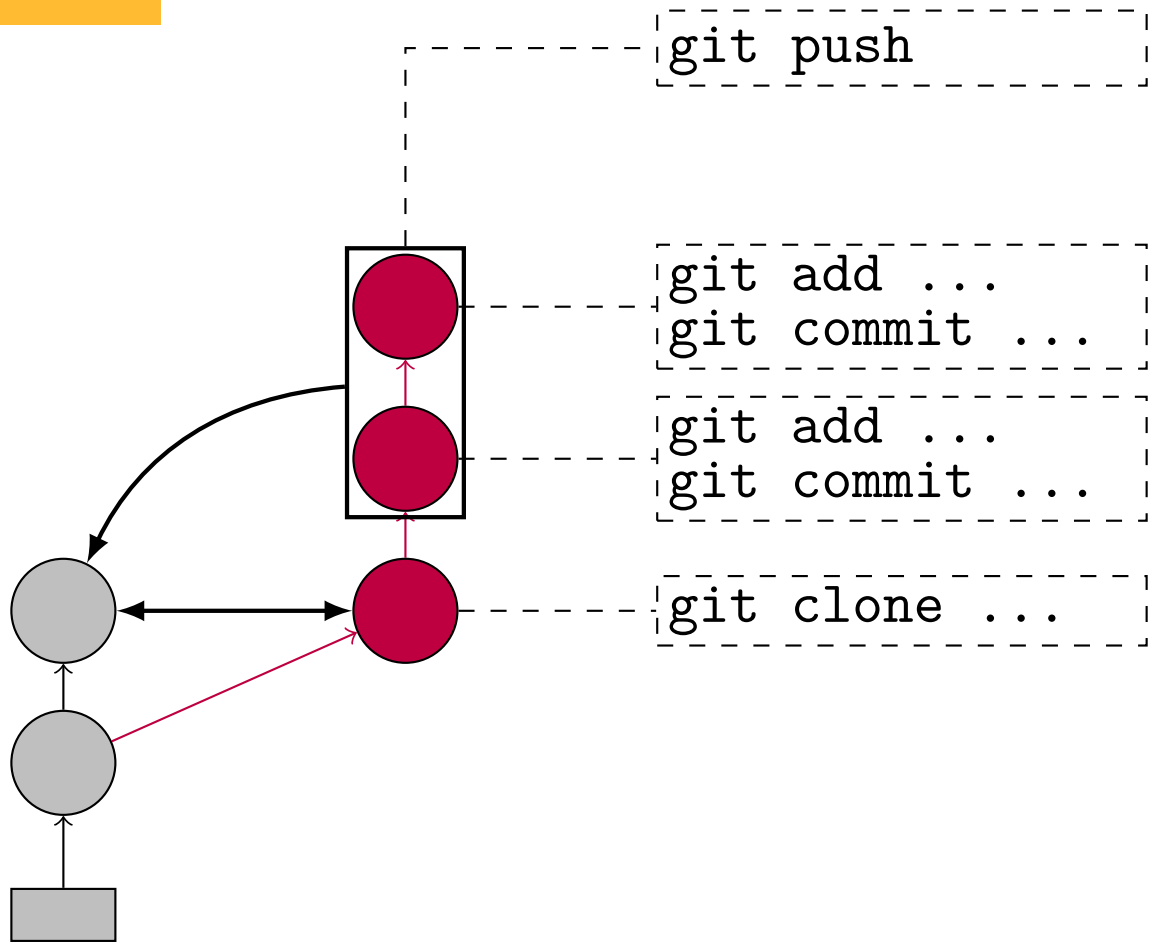


---

versions : sur dépôt distant – sur dépôt local

On clone le dépôt, et on travaille sur le projet.

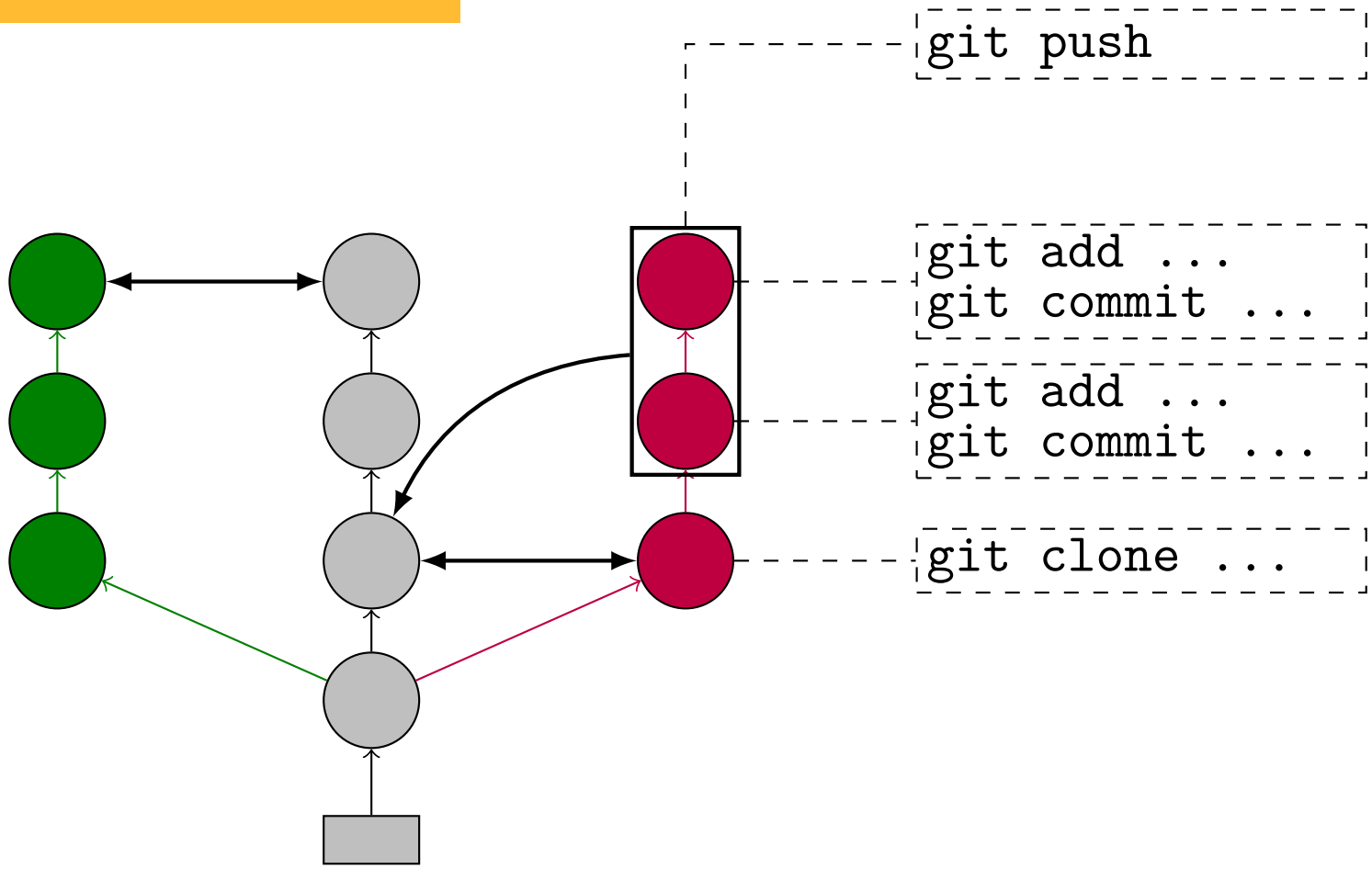
git pull avec fusion



versions : sur dépôt distant – sur dépôt local

Après quelques heures de travail, on pousse nos versions sur le dépôt distant.

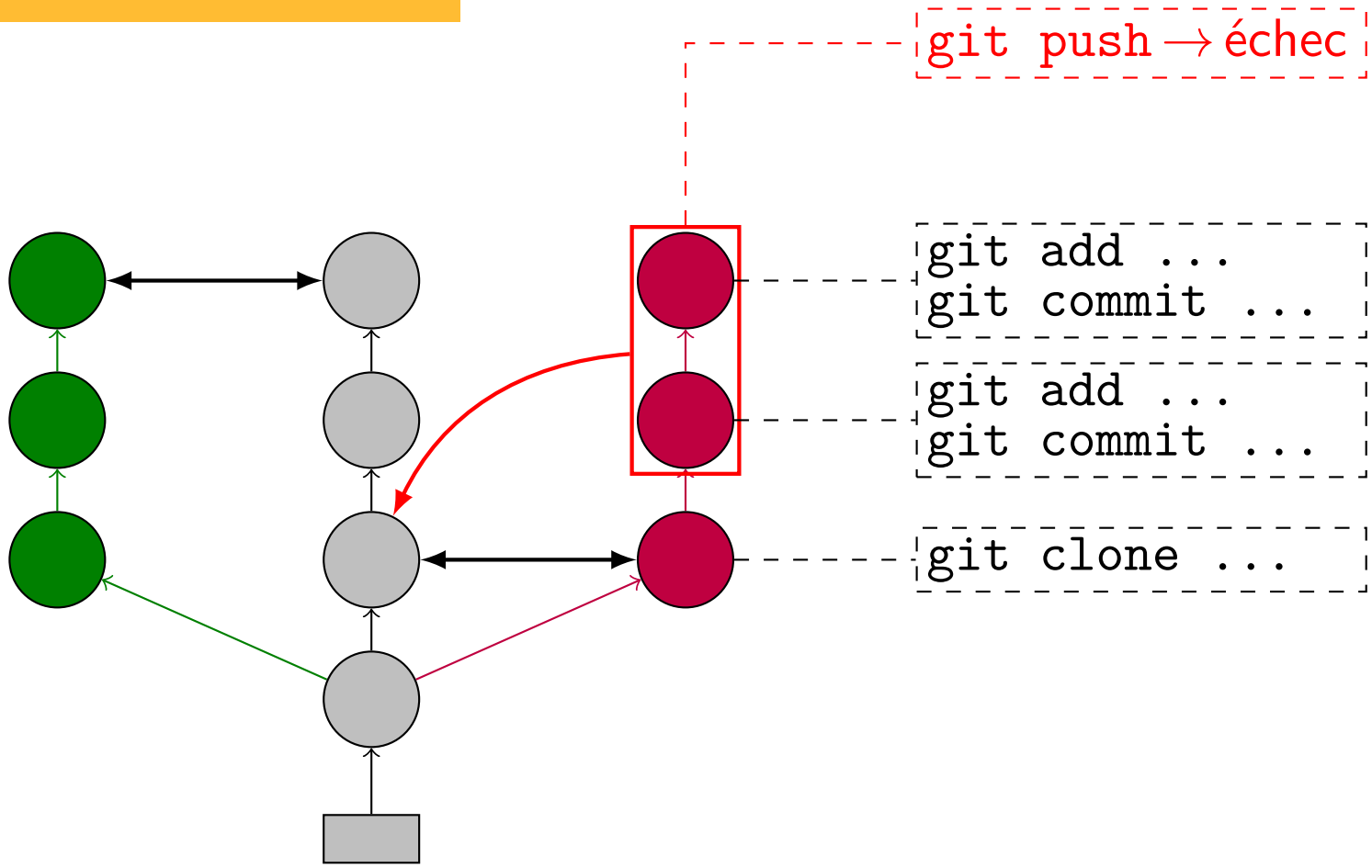
git pull avec fusion



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Après quelques heures de travail, on pousse nos versions sur le dépôt distant.
Mais un collaborateur a poussé des versions entre temps.

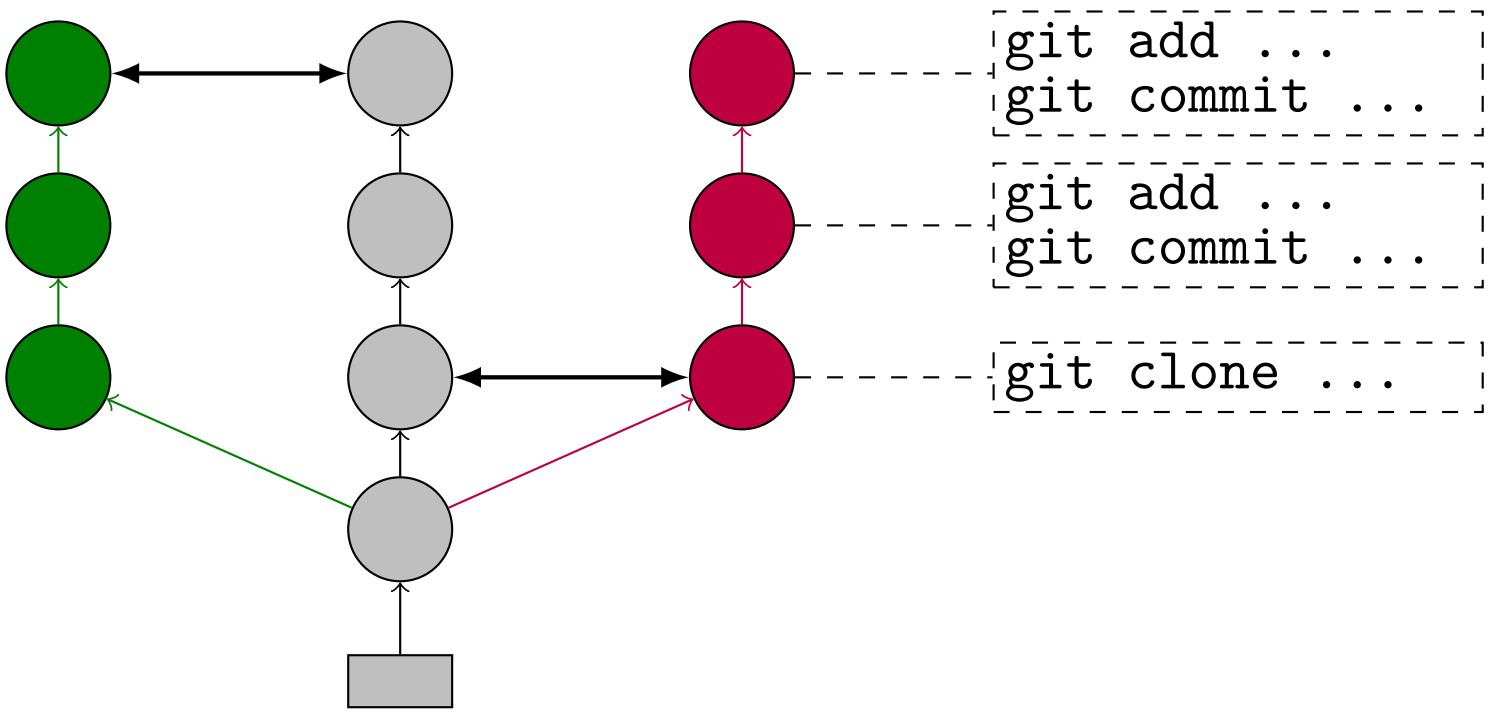
git pull avec fusion



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Après quelques heures de travail, on pousse nos versions sur le dépôt distant.
Mais un collaborateur a poussé des versions entre temps.
Le `git push` échoue.

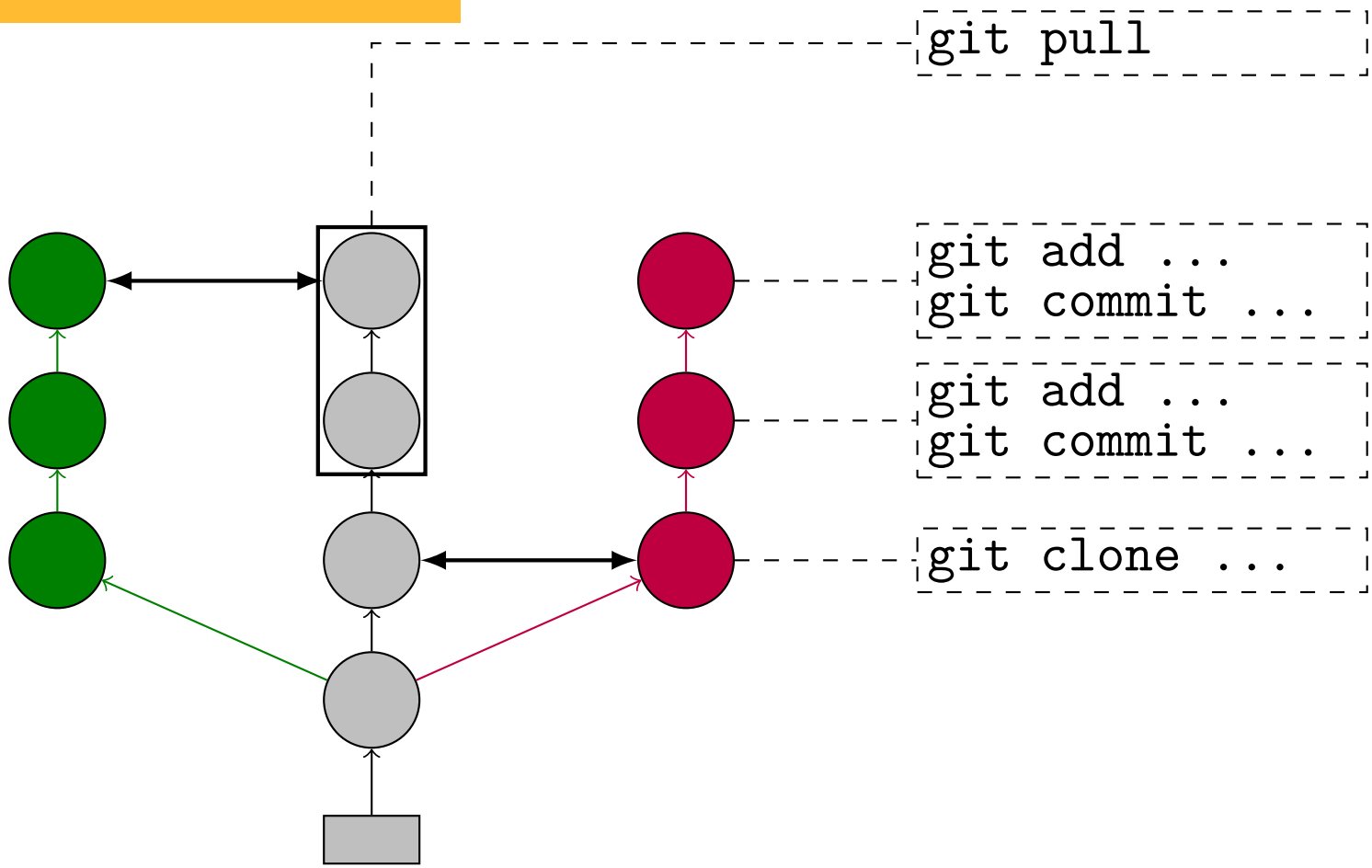
`git pull` avec fusion



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Après quelques heures de travail, on pousse nos versions sur le dépôt distant.
Mais un collaborateur a poussé des versions entre temps.
Le `git push` échoue – il faut d'abort faire `git pull`.

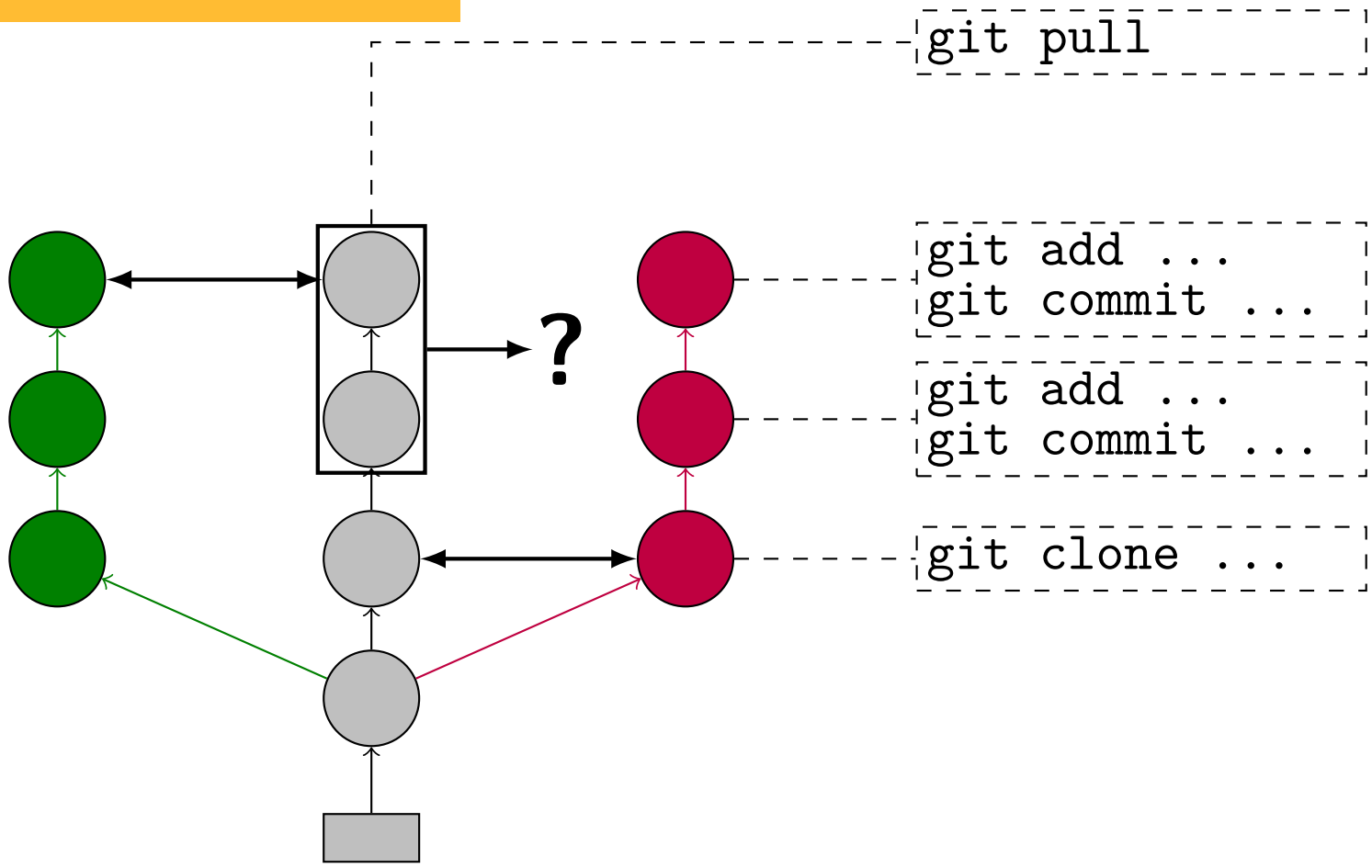
`git pull` avec fusion



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Nous faisons `git pull`, mais que se passe-t-il ?

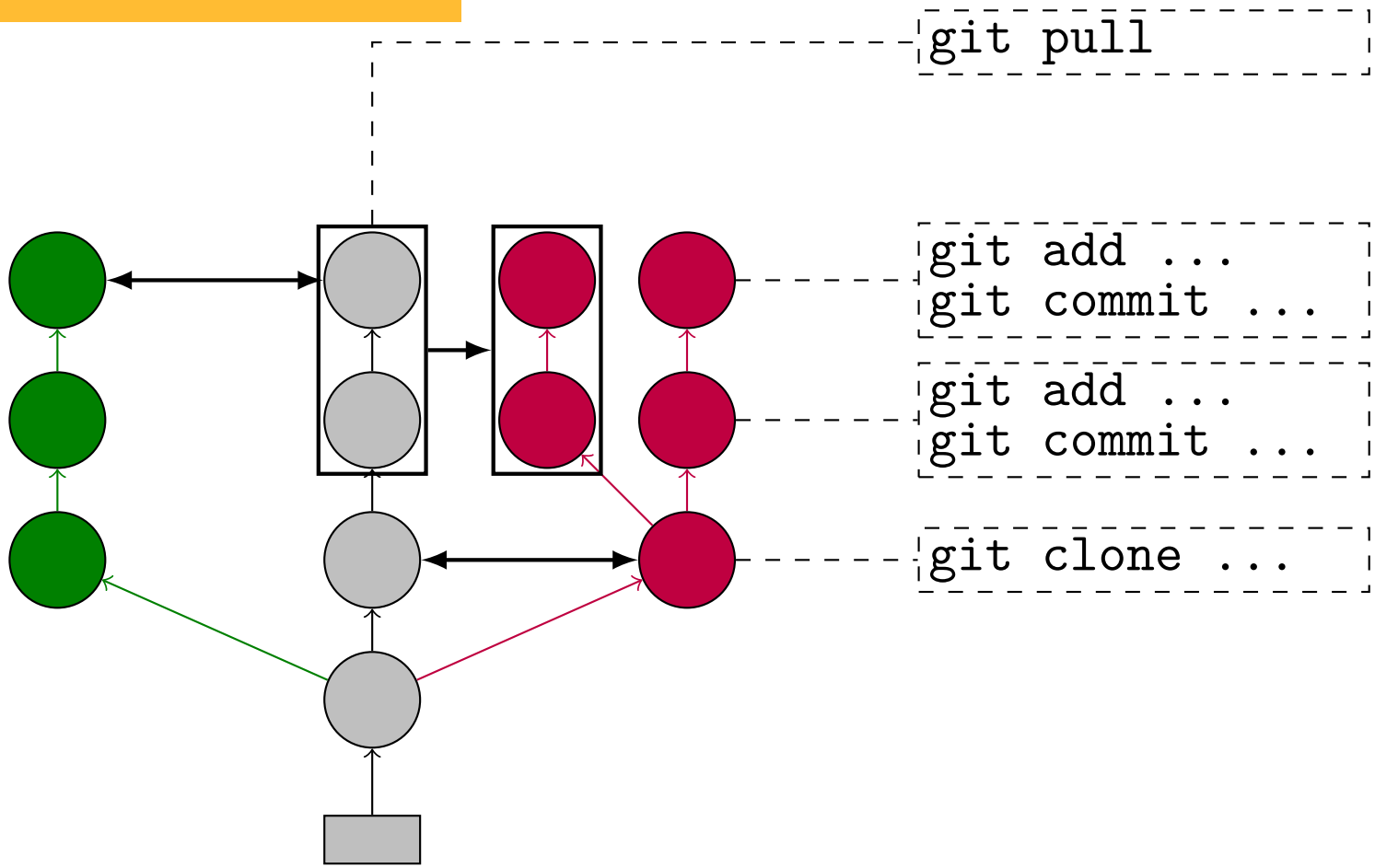
`git pull` avec fusion



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Nous faisons `git pull`, mais que se passe-t-il ?

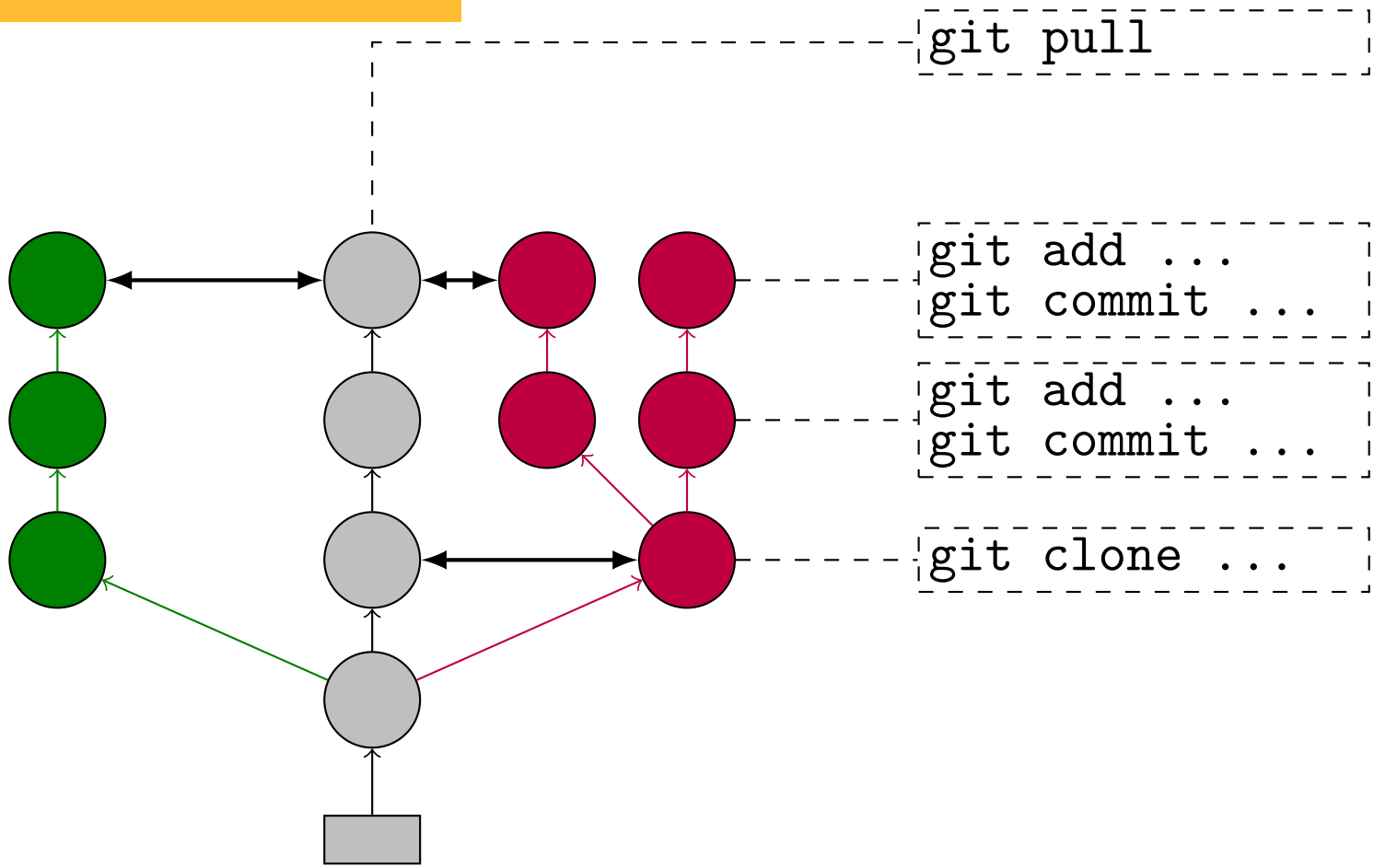
`git pull` avec fusion



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Nous faisons `git pull`, mais que se passe-t-il ?

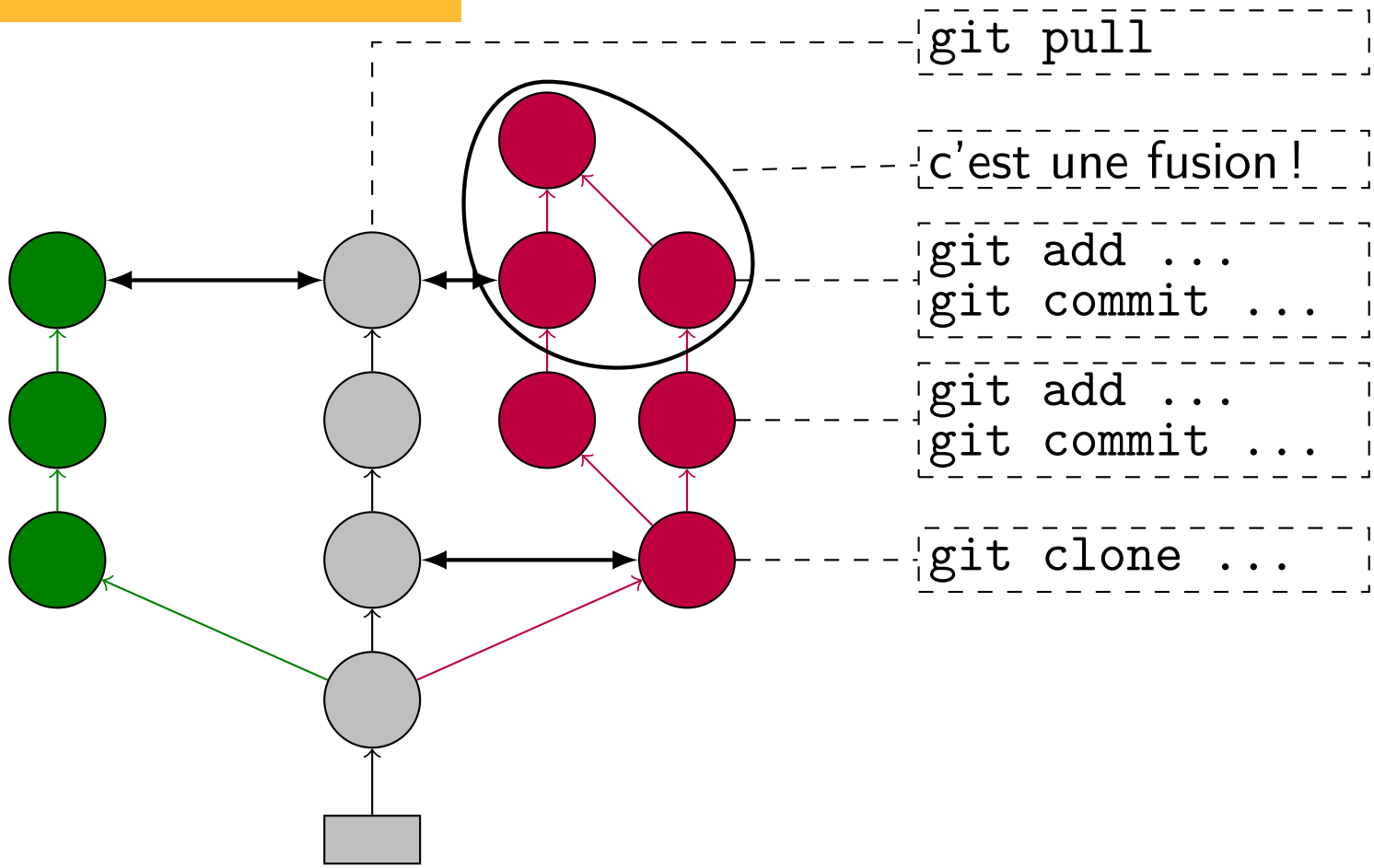
`git pull` avec fusion



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Nous faisons `git pull`, mais que se passe-t-il ?

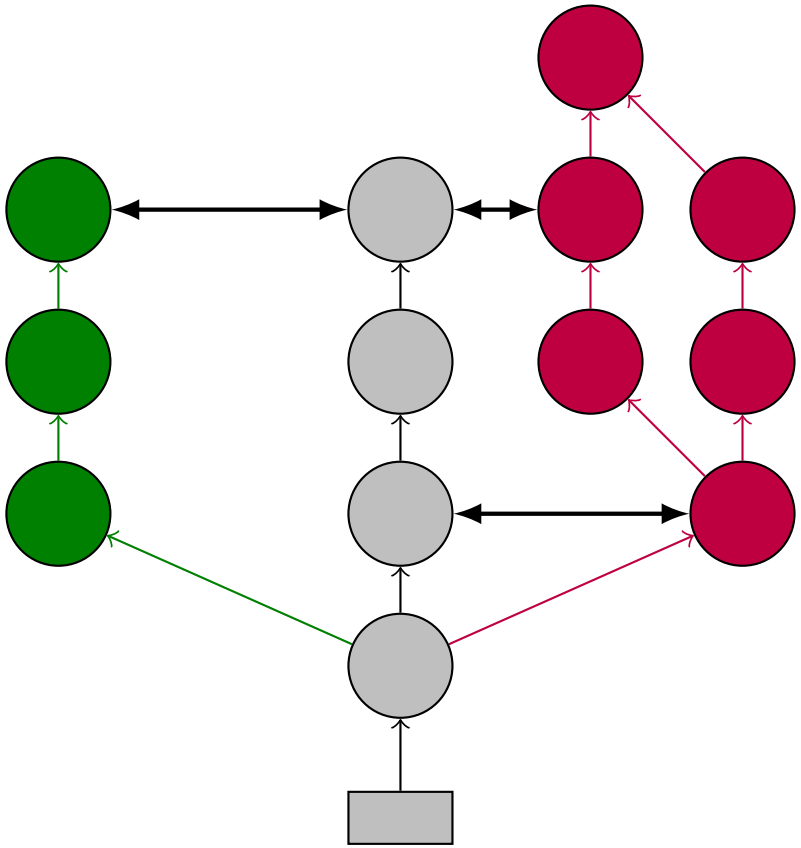
`git pull` avec fusion



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

Nous faisons `git pull`, mais que se passe-t-il ?
Une nouvelle version est créée : fusion des deux branches.

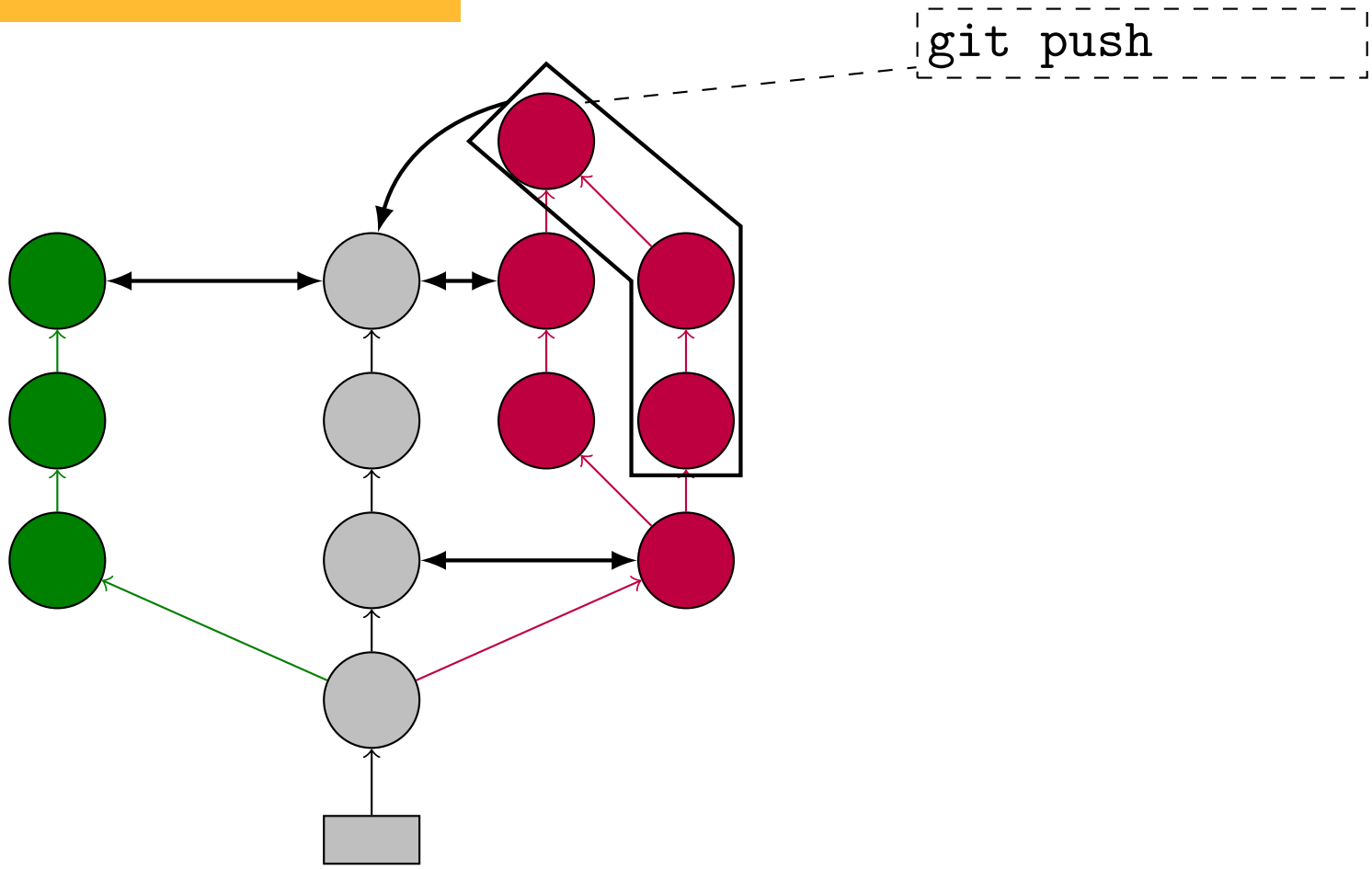
`git pull` avec fusion



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

La version fusionnée est locale, mais on peut relancer notre `git push`.

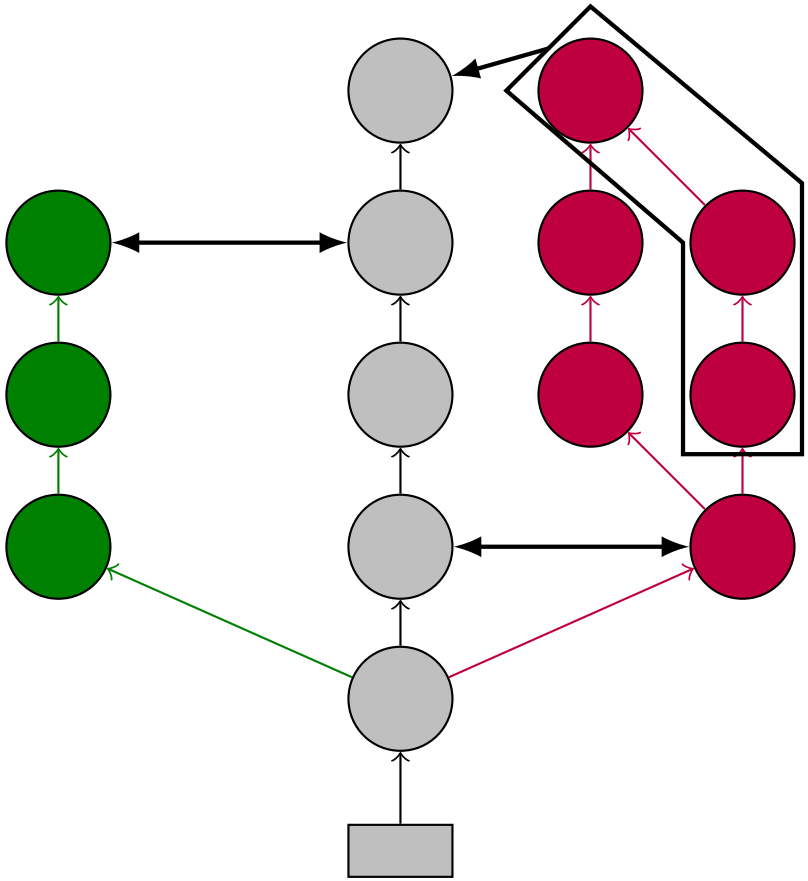
`git pull` avec fusion



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

La version fusionnée est locale, mais on peut relancer notre `git push`.

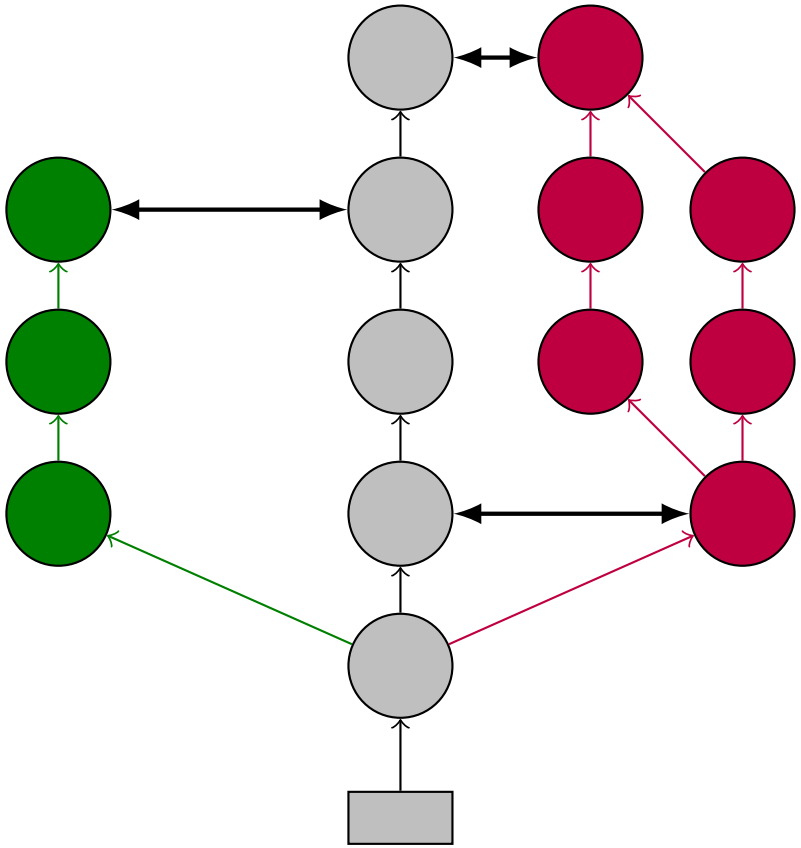
`git pull` avec fusion



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

La version fusionnée est locale, mais on peut relancer notre `git push`.

`git pull` avec fusion



versions : sur dépôt distant – sur dépôt local – d'un collaborateur

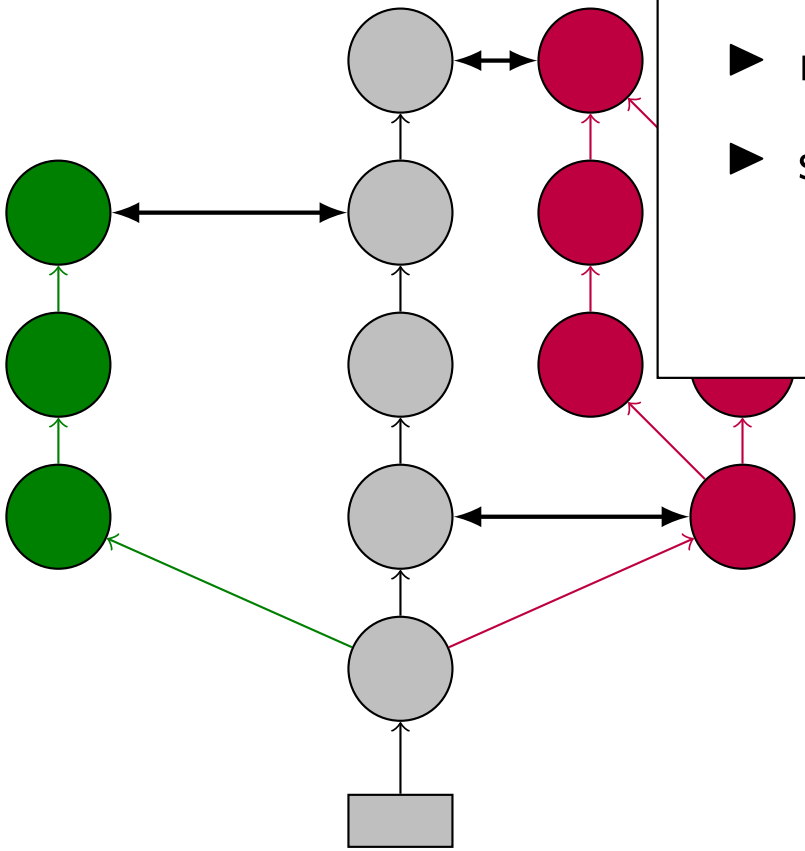
La version fusionnée est locale, mais on peut relancer notre `git push`.

`git pull` avec fusion

`git` gère au mieux les fusions :

- modifs sur fichiers différents : ok
- modifs sur lignes différentes : ok
- sinon :

  besoin d'une intervention humaine.



versions : ● sur dépôt distant – ● sur dépôt local – ● d'un collaborateur

La version fusionnée est locale, mais on peut relancer notre `git push`.

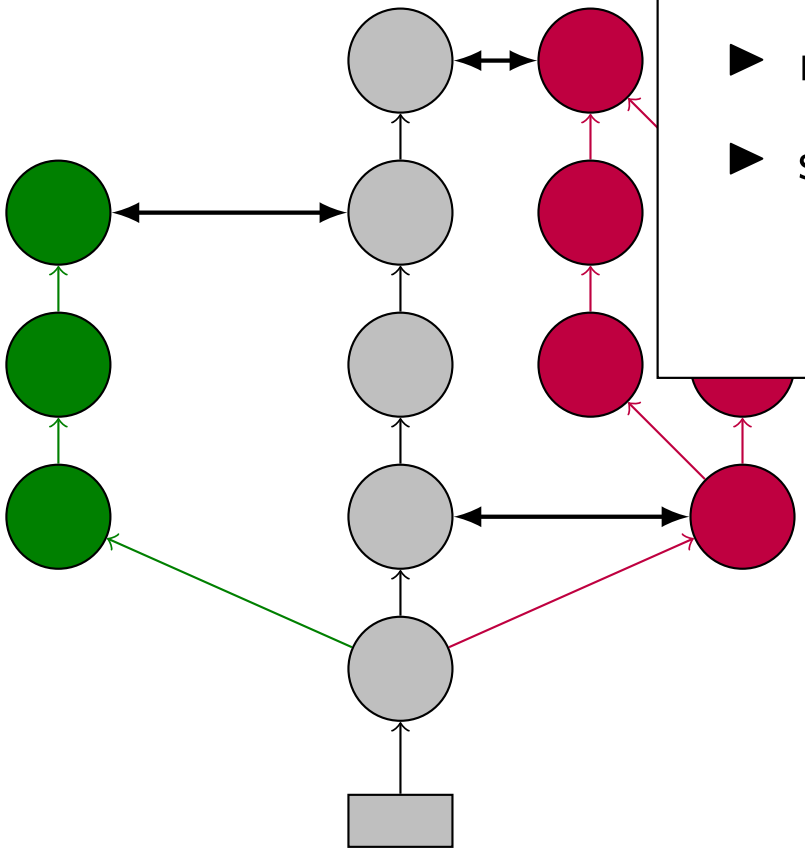
`git pull` avec fusion

`git` gère au mieux les fusions :

- modifs sur fichiers différents : ok
- modifs sur lignes différentes : ok
- sinon :

besoin d'une intervention humaine.

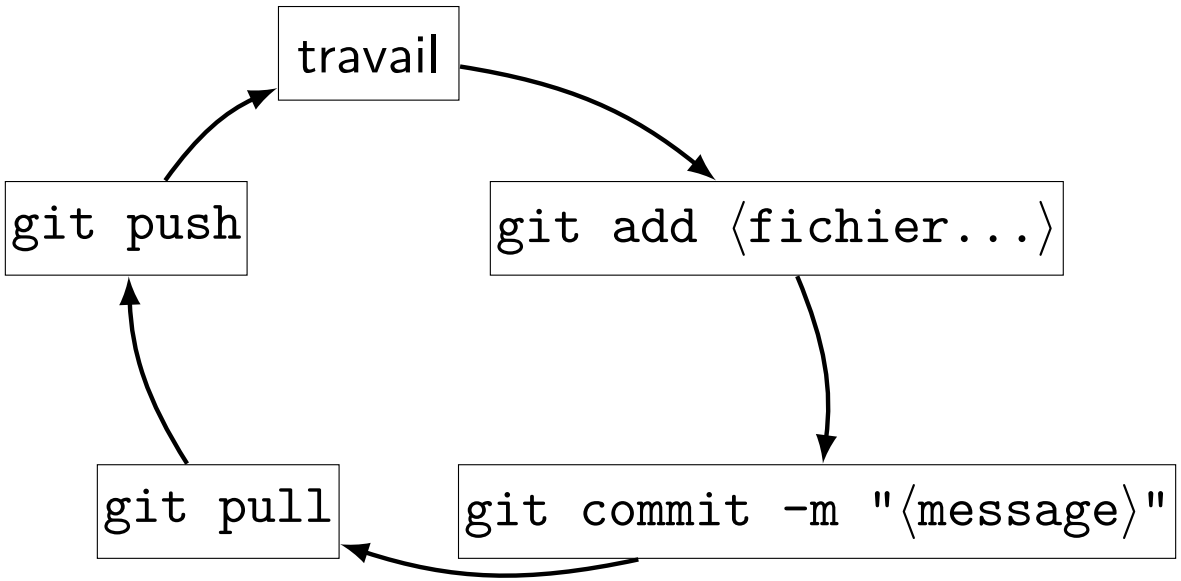
c.f., prochains cours/TP...



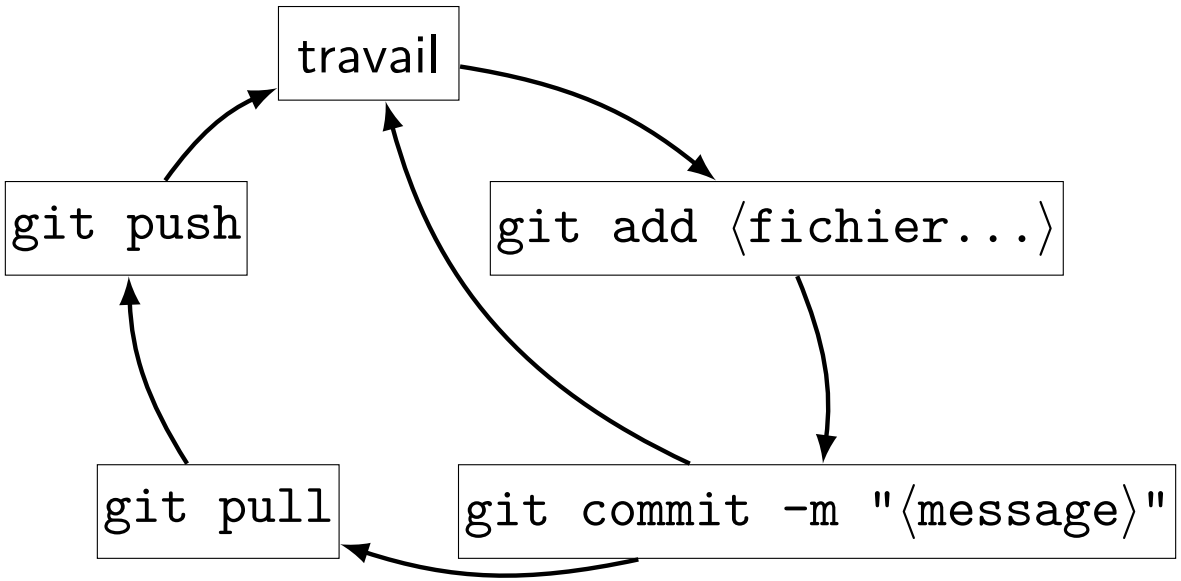
versions : sur dépôt distant – sur dépôt local – d'un collaborateur

La version fusionnée est locale, mais on peut relancer notre `git push`.

Utilisation standard (collaboration)

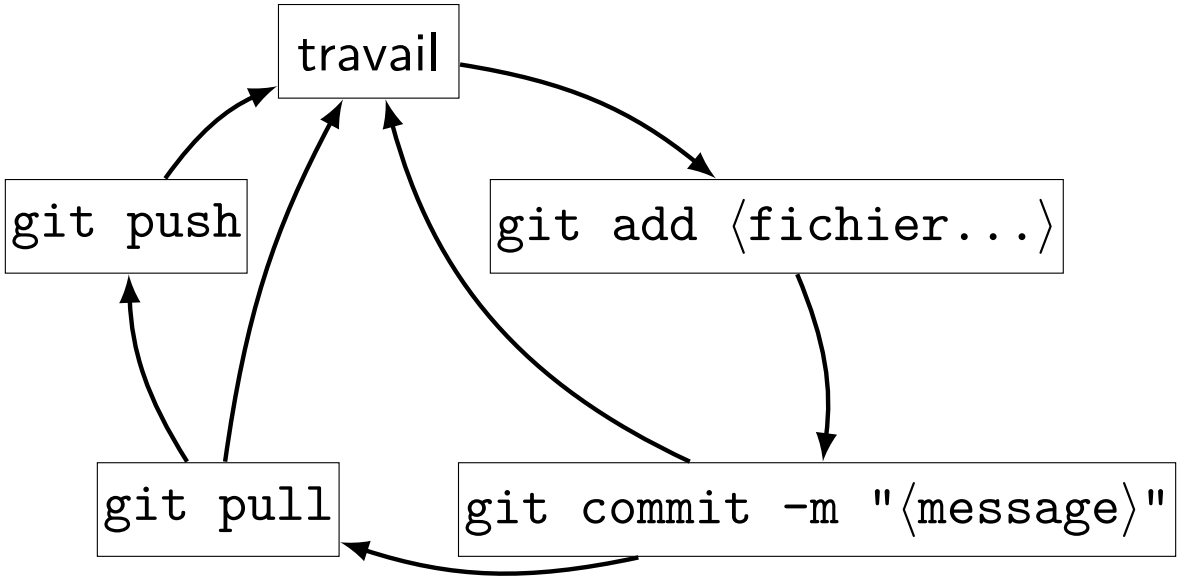


Utilisation standard (collaboration)



- Il n'est pas nécessaire de se synchroniser à chaque version :
  mieux vaut attendre d'avoir fini la session de travail.

Utilisation standard (collaboration)



- Il n'est pas nécessaire de se synchroniser à chaque version :
  mieux vaut attendre d'avoir fini la session de travail.
- Récupérer les versions des autres régulièrement est une bonne idée :
  cela permet de réaliser le travail de fusion au fur et à mesure.

Bonnes pratiques

- Faire des versions régulièrement avec des messages informatifs.
- Récupérer les versions depuis le dépôt assez régulièrement pour éviter à avoir à faire une fusion manuelle compliquée.
- Diviser les projets en plusieurs fichiers.
- Écrire des lignes courtes.
- Pousser son travail quand une version stable a été obtenue ou avant une interruption longue de travail.


LATEX

Retour sur les tableaux

Les tableaux

L'environnement `tabular` permet de créer des tableaux.

```
\begin{tabular}{lcr}

\end{tabular}
```

Les tableaux

L'environnement `tabular` permet de créer des tableaux.

- avec un certain nombre de colonnes fixé

```
\begin{tabular}{lcr}

\end{tabular}
```

Les tableaux

L'environnement `tabular` permet de créer des tableaux.

- avec un certain nombre de colonnes fixé
- et un nombre de lignes variables

```
\begin{tabular}{lcr}

\end{tabular}
```

Les tableaux

L'environnement `tabular` permet de créer des tableaux.

- avec un certain nombre de colonnes fixé
  spécifié par l'argument positionnel de l'environnement
- et un nombre de lignes variables

```
\begin{tabular}{lcr}



\end{tabular}
```

Les tableaux

L'environnement `tabular` permet de créer des tableaux.

- avec un certain nombre de colonnes fixé
  spécifié par l'argument positionnel de l'environnement
  pour chaque colonne, on indique l'alignement de texte (***l**eft*, ***c**entered*, ***r**ight*)
- et un nombre de lignes variables

```
\begin{tabular}{lcr}



\end{tabular}
```

Les tableaux

L'environnement `tabular` permet de créer des tableaux.

- avec un certain nombre de colonnes fixé
  spécifié par l'argument positionnel de l'environnement
  pour chaque colonne, on indique l'alignement de texte (***l**eft*, ***c**entered*, ***r**ight*)
- et un nombre de lignes variables

```
\begin{tabular}{lcr}



\end{tabular}
```

Les tableaux

L'environnement `tabular` permet de créer des tableaux.

- avec un certain nombre de colonnes fixé
  spécifié par l'argument positionnel de l'environnement
  pour chaque colonne, on indique l'alignement de texte (*left*, *centered*, *right*)
- et un nombre de lignes variables

```
\begin{tabular}{lcr}



\end{tabular}
```

Les tableaux

L'environnement `tabular` permet de créer des tableaux.

- avec un certain nombre de colonnes fixé
  spécifié par l'argument positionnel de l'environnement
  pour chaque colonne, on indique l'alignement de texte (***l**eft*, ***c**entered*, ***r**ight*)
- et un nombre de lignes variables

```
\begin{tabular}{lcr}



\end{tabular}
```

Les tableaux

L'environnement `tabular` permet de créer des tableaux.

- avec un certain nombre de colonnes fixé
  spécifié par l'argument positionnel de l'environnement
  pour chaque colonne, on indique l'alignement de texte (***l**eft*, ***c**entered*, ***r**ight*)
- et un nombre de lignes variables

Le corps (*body*) de l'environnement donne le contenu du tableau

---

```
\begin{tabular}{lcr}

    <BODY>

\end{tabular}
```

Les tableaux

L'environnement `tabular` permet de créer des tableaux.

- avec un certain nombre de colonnes fixé
  spécifié par l'argument positionnel de l'environnement
  pour chaque colonne, on indique l'alignement de texte (***l**eft*, ***c**entered*, ***r**ight*)
- et un nombre de lignes variables

Le corps (*body*) de l'environnement donne le contenu du tableau

- ligne par ligne, séparées par `\\`

```
\begin{tabular}{lcr}
  <première ligne>      \\
  <deuxième ligne>      \\
  <troisième ligne>     \\
\end{tabular}
```

Les tableaux

L'environnement `tabular` permet de créer des tableaux.

- avec un certain nombre de colonnes fixé
  spécifié par l'argument positionnel de l'environnement
  pour chaque colonne, on indique l'alignement de texte (***l**eft*, ***c**entered*, ***r**ight*)
- et un nombre de lignes variables

Le corps (*body*) de l'environnement donne le contenu du tableau

- ligne par ligne, séparées par `\\`
  - et colonne par colonne, séparées par `&`

```
\begin{tabular}{lcr}
  a0 & b0 & c0          \\
  aa1 & bb1 & cc1       \\
  aaa2 & bbb2 & ccc2    \\
\end{tabular}
```

Les tableaux

L'environnement `tabular` permet de créer des tableaux.

- avec un certain nombre de colonnes fixé
  spécifié par l'argument positionnel de l'environnement
  pour chaque colonne, on indique l'alignement de texte (***l**eft*, ***c**entered*, ***r**ight*)
- et un nombre de lignes variables

Le corps (*body*) de l'environnement donne le contenu du tableau

- ligne par ligne, séparées par `\\`
  - et colonne par colonne, séparées par `&`

```
\begin{tabular}{lcr}
  a0 & b0 & c0          \\
  aa1 & bb1 & cc1       \\
  aaa2 & bbb2 & ccc2    \\
\end{tabular}
```

a0	b0	c0
aa1	bb1	cc1
aaa2	bbb2	ccc2

Bordures

Les tableaux n'ont par défaut pas de bordures.

```
\begin{tabular}{lcr}
    a0 & b0 & c0 \\
    aa1 & bb1 & cc1 \\
    aaa2 & bbb2 & ccc2

\end{tabular}
```

Bordures

Les tableaux n'ont par défaut pas de bordures.

```
\begin{tabular}{lcr}
    a0 & b0 & c0 \\
    aa1 & bb1 & cc1 \\
    aaa2 & bbb2 & ccc2

\end{tabular}
```

a0	b0	c0
aa1	bb1	cc1
aaa2	bbb2	ccc2

Bordures

Les tableaux n'ont par défaut pas de bordures.

- bordures **verticales** en ajoutant | dans les spécifications de colonnes

```
\begin{tabular}{l|cr}
    a0 & b0 & c0 \\
    aa1 & bb1 & cc1 \\
    aaa2 & bbb2 & ccc2

\end{tabular}
```

a0	b0	c0
aa1	bb1	cc1
aaa2	bbb2	ccc2

Bordures

Les tableaux n'ont par défaut pas de bordures.

- bordures **verticales** en ajoutant | dans les spécifications de colonnes

```
\begin{tabular}{|l|cr|}
    a0 & b0 & c0 \\
    aa1 & bb1 & cc1 \\
    aaa2 & bbb2 & ccc2

\end{tabular}
```

a0	b0	c0
aa1	bb1	cc1
aaa2	bbb2	ccc2

Bordures

Les tableaux n'ont par défaut pas de bordures.

- bordures **verticales** en ajoutant | dans les spécifications de colonnes

```
\begin{tabular}{|l||c|r|}
    a0 & b0 & c0 \\
    aa1 & bb1 & cc1 \\
    aaa2 & bbb2 & ccc2

\end{tabular}
```

a0	b0	c0
aa1	bb1	cc1
aaa2	bbb2	ccc2

Bordures

Les tableaux n'ont par défaut pas de bordures.

- bordures **verticales** en ajoutant | dans les spécifications de colonnes
- bordures **horizontales** en ajoutant \hline en début de ligne

```
\begin{tabular}{|l||c|r|}
   a0 & b0 & c0 \\
  \hline aa1 & bb1 & cc1 \\
   aaa2 & bbb2 & ccc2

\end{tabular}
```

a0	b0	c0
aa1	bb1	cc1
aaa2	bbb2	ccc2

Bordures

Les tableaux n'ont par défaut pas de bordures.

- bordures **verticales** en ajoutant | dans les spécifications de colonnes
- bordures **horizontales** en ajoutant \hline en début de ligne

```
\begin{tabular}{|l||c|r|}
  \hline a0 & b0 & c0 \\
  \hline aa1 & bb1 & cc1 \\
   aaa2 & bbb2 & ccc2 \\
  \hline
\end{tabular}
```

a0	b0	c0
aa1	bb1	cc1
aaa2	bbb2	ccc2

Bordures

Les tableaux n'ont par défaut pas de bordures.

- bordures **verticales** en ajoutant | dans les spécifications de colonnes
- bordures **horizontales** en ajoutant \hline en début de ligne

```
\begin{tabular}{|l||c|r|}
  \hline a0 & b0 & c0 \\
  \hline\hline aa1 & bb1 & cc1 \\
  \hline aaa2 & bbb2 & ccc2 \\
  \hline
\end{tabular}
```

a0	b0	c0
aa1	bb1	cc1
aaa2	bbb2	ccc2

Fusions de cellules sur une même ligne

La commande `\multicolumn` permet de fusionner
des cellules consécutives d'une même ligne.

Fusions de cellules sur une même ligne

La commande `\multicolumn` permet de fusionner des cellules consécutives d'une même ligne.

3 arguments : `\multicolumn{nombre}{alignement}{texte}`

Fusions de cellules sur une même ligne

La commande `\multicolumn` permet de fusionner des cellules consécutives d'une même ligne.

3 arguments : `\multicolumn{nombre}{alignement}{texte}`

```
\begin{tabular}{|l|c|r|}
  \hline a0 & b0 & c0 \\
  \hline
     aa1 & bb1
    & cc1 \\
  \hline aaa2 & bbb2 & ccc2 \\
  \hline
\end{tabular}
```

a0	b0	c0
aa1	bb1	cc1
aaa2	bbb2	ccc2

Fusions de cellules sur une même ligne

La commande \multicolumn permet de fusionner des cellules consécutives d'une même ligne.

3 arguments : \multicolumn{nombre}{alignement}{texte}

```
\begin{tabular}{|l|c|r|}
  \hline a0 & b0 & c0 \\
  \hline
    \multicolumn{2}{r}{aa1bb1}
    & cc1 \\
  \hline aaa2 & bbb2 & ccc2 \\
  \hline
\end{tabular}
```

a0	b0	c0
	aa1bb1	cc1
aaa2	bbb2	ccc2

Fusions de cellules sur une même ligne

La commande \multicolumn permet de fusionner des cellules consécutives d'une même ligne.

3 arguments : \multicolumn{nombre}{alignement}{texte}

```
\begin{tabular}{|l|c|r|}
  \hline a0 & b0 & c0 \\
  \hline
    \multicolumn{2}{r}{aa1bb1}
    & cc1 \\
  \hline aaa2 & bbb2 & ccc2 \\
  \hline
\end{tabular}
```

a0	b0	c0
	aa1bb1	cc1
aaa2	bbb2	ccc2

Oh non, mes jolies bordures ! ! !

Fusions de cellules sur une même ligne

La commande `\multicolumn` permet de fusionner des cellules consécutives d'une même ligne.

3 arguments : `\multicolumn{nombre}{alignement}{texte}`

```
\begin{tabular}{|l|c|r|}
  \hline a0 & b0 & c0 \\
  \hline
    \multicolumn{2}{|r|}{aa1bb1}
    & cc1 \\
  \hline aaa2 & bbb2 & ccc2 \\
  \hline
\end{tabular}
```

a0	b0	c0
aa1bb1		cc1
aaa2	bbb2	ccc2

Oh non, mes jolies bordures ! ! ! Ouf.

Aller plus loin sur les tableaux

Le paquet `multirow` définit une commande `\multirow`
qui permet de fusionner des cellules verticalement.

Aller plus loin sur les tableaux

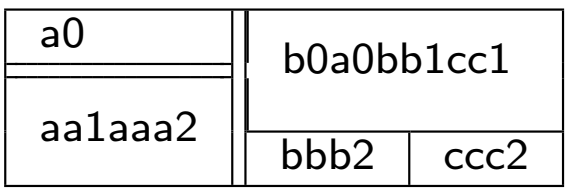
Le paquet `multirow` définit une commande `\multirow`
qui permet de fusionner des cellules verticalement.

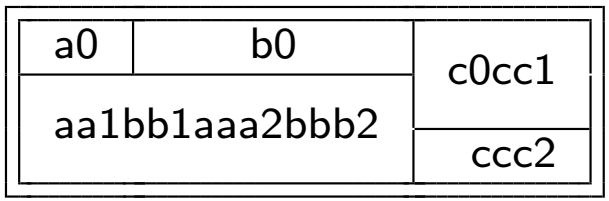
Le paquet `hhline` définit `\hhline`, une variante de `\hline`,
qui gère joliment les intersections.

Aller plus loin sur les tableaux

Le paquet `multirow` définit une commande `\multirow`
qui permet de fusionner des cellules verticalement.

Le paquet `hhline` définit `\hhline`, une variante de `\hline`,
qui gère joliment les intersections.

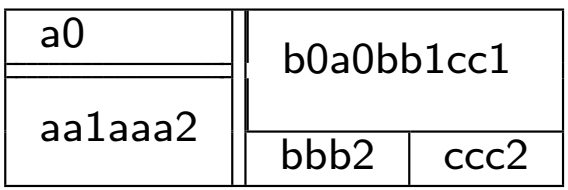


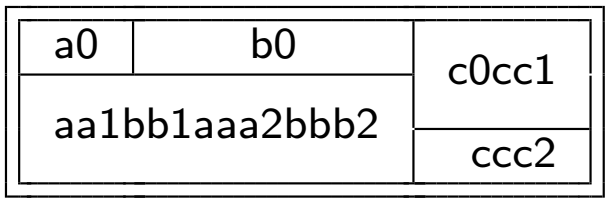


Aller plus loin sur les tableaux

Le paquet `multirow` définit une commande `\multirow`
qui permet de fusionner des cellules verticalement.

Le paquet `hhline` définit `\hhline`, une variante de `\hline`,
qui gère joliment les intersections.



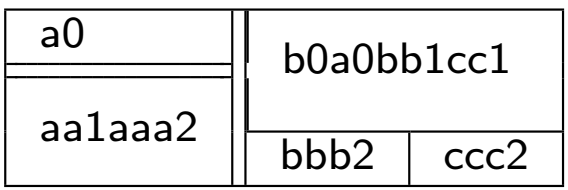


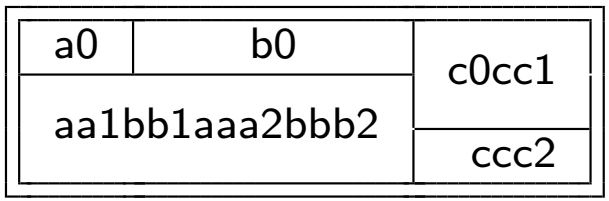
Les paquets `tabular*`, `tabularx`, `tabulary`, `array`
permettent de généraliser les spécifications de colonne.

Aller plus loin sur les tableaux

Le paquet `multirow` définit une commande `\multirow`
qui permet de fusionner des cellules verticalement.

Le paquet `hhline` définit `\hhline`, une variante de `\hline`,
qui gère joliment les intersections.





Les paquets `tabular*`, `tabularx`, `tabulary`, `array`
permettent de généraliser les spécifications de colonne.

Le paquet `makecell` permet de définir des styles,
notamment pour les entêtes de tableaux.

Aller plus loin sur les tableaux

Le paquet `multirow` définit une commande `\multirow`
qui permet de fusionner des cellules verticalement.

Le paquet `hhline` définit `\hhline`, une variante de `\hline`,
qui gère joliment les intersections.

a0	b0a0bb1cc1	
aa1aaa2	bbb2	ccc2

a0	b0	c0cc1
aa1bb1aaa2bbb2		ccc2

Les paquets `tabular*`, `tabularx`, `tabulary`, `array`
permettent de généraliser les spécifications de colonne.

Le paquet `makecell` permet de définir des styles,
notamment pour les entêtes de tableaux.

Le magnifique et moderne paquet `tabularray`
reprend, améliore et complète toutes les bonnes idées des précédents.

Inclure du contenu LaTeX depuis un autre fichier

Inclure du contenu depuis un autre fichier

La commande `\input{chemin-vers-fichier}`

inclut le contenu du fichier indiqué par chemin-vers-fichier

dans le code LaTeX (à l'endroit où est placé la commande)

Inclure du contenu depuis un autre fichier

La commande `\input{chemin-vers-fichier}`
inclut le contenu du fichier indiqué par chemin-vers-fichier
dans le code LaTeX (à l'endroit où est placé la commande)

subfile.tex

```
blabla


blabla
```

mainfile.tex

```
bla
\input{subfile}
bla
```

≡

"ce que voit LaTeX"

```
bla
blabla


blabla
bla
```

Inclure du contenu depuis un autre fichier

La commande `\input{chemin-vers-fichier}`
inclut le contenu du fichier indiqué par chemin-vers-fichier
dans le code LaTeX (à l'endroit où est placé la commande)

subfile.tex

```
blabla


blabla
```

mainfile.tex

```
bla
\input{subfile}
bla
```

≡

"ce que voit LaTeX"

```
bla
blabla


blabla
bla
```

But :

- isoler des parties complexes de code
- structurer le code (*e.g.*, chaque section dans un fichier différent)
- réutiliser des morceaux de code dans différents document
- collaborer efficacement (fusions automatique assurées avec git)

Exemple

```
\documentclass[french]{article}
\input{./common-packages}

\title{Document réunissant d'autres documents}
\author{Bruno GUILLON}
\date{Novembre 2024}

\begin{document}

\section{Définitions}
\label{sec/def}
\input{./sections/definitions}

\section{Théorèmes}
\label{sec/thm}
\input{./sections/theorems}

\end{document}
```

Aller plus loin sur l'inclusion de source LaTeX

La commande `\include` est une variation de `\input`

utile surtout pour de gros projets (*e.g.*, chapitres de livre).

Aller plus loin sur l'inclusion de source LaTeX

La commande `\include` est une variation de `\input`
utile surtout pour de gros projets (*e.g.*, chapitres de livre).

La classe `standalone` permet de donner un préambule aux fichiers à inclure
ignoré par la commande `\includestandalone` du paquet `standalone`

Aller plus loin sur l'inclusion de source LaTeX

La commande `\include` est une variation de `\input`
utile surtout pour de gros projets (*e.g.*, chapitres de livre).

La classe `standalone` permet de donner un préambule aux fichiers à inclure
ignoré par la commande `\includestandalone` du paquet `standalone`
+ options sympathiques, *e.g.*, document pré-compilé...

Aller plus loin sur l'inclusion de source LaTeX

La commande `\include` est une variation de `\input`
utile surtout pour de gros projets (*e.g.*, chapitres de livre).

La classe `standalone` permet de donner un préambule aux fichiers à inclure
ignoré par la commande `\includestandalone` du paquet `standalone`
+ options sympathiques, *e.g.*, document pré-compilé...

Le paquet `xr` permet de faire des références (*via* `\ref`)
à des étiquettes définies (*via* `\label`) dans d'autres fichiers

Insérer des images

Insérer une image

La commande `\includegraphics` du paquet `graphicx`[1]
permet d'inclure une image là où elle est appelée

1. amélioration du paquet `graphics`

Insérer une image

La commande `\includegraphics` du paquet `graphicx`[1]
permet d'inclure une image là où elle est appelée

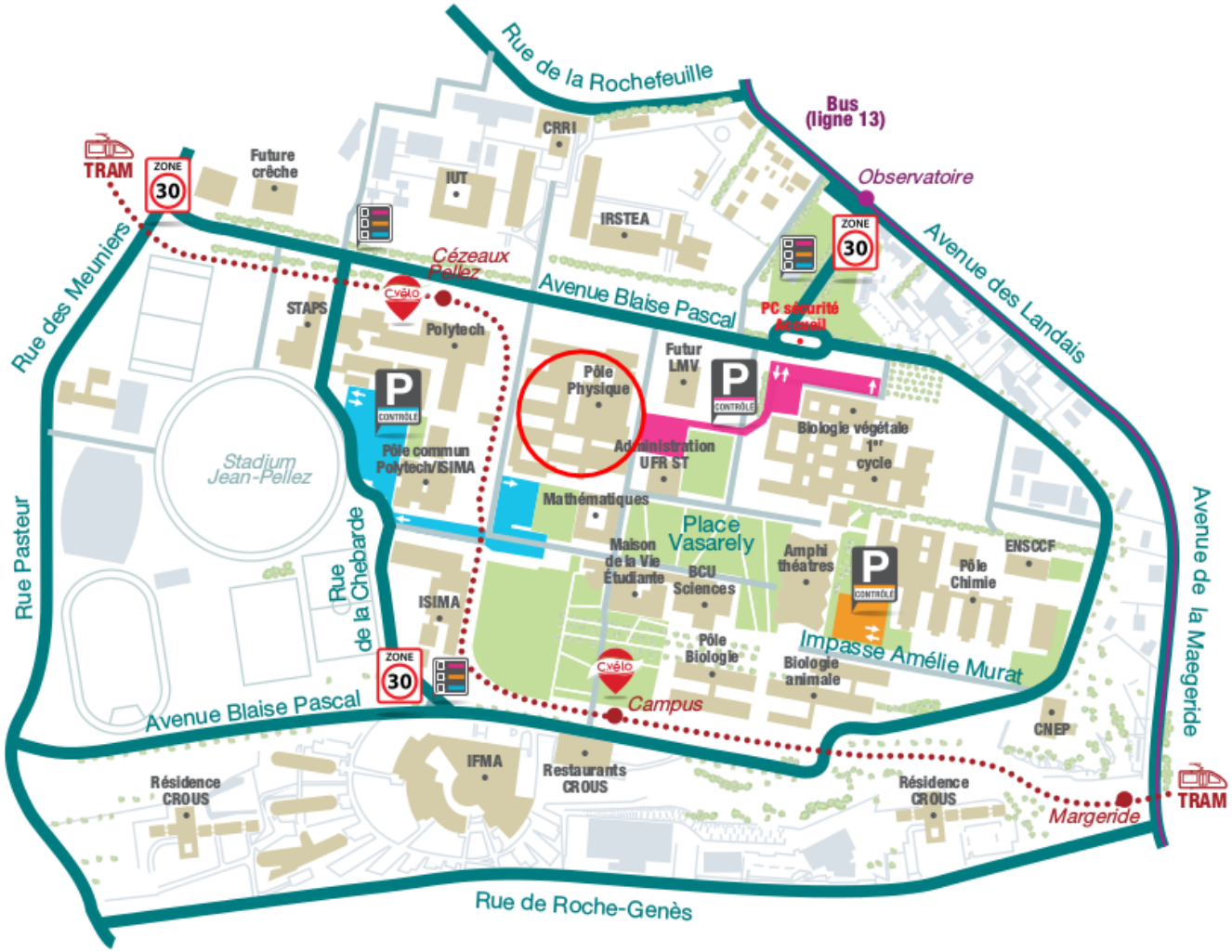
Par exemple :

```
J'\includegraphics{pics/heart.png}
LaTeX, pas vous?
```

J'  LaTeX, pas vous ?

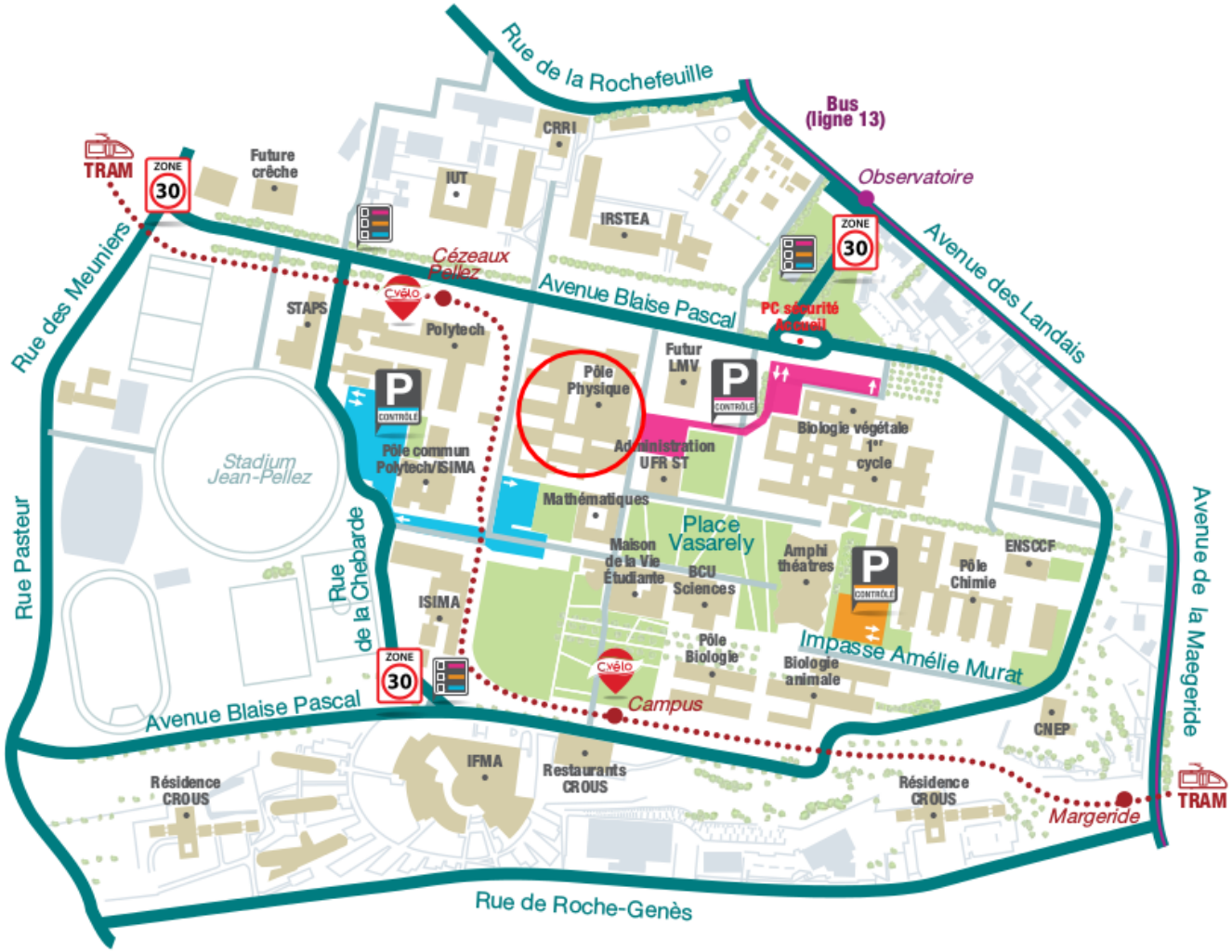
1. amélioration du paquet `graphics`

Vous êtes ici



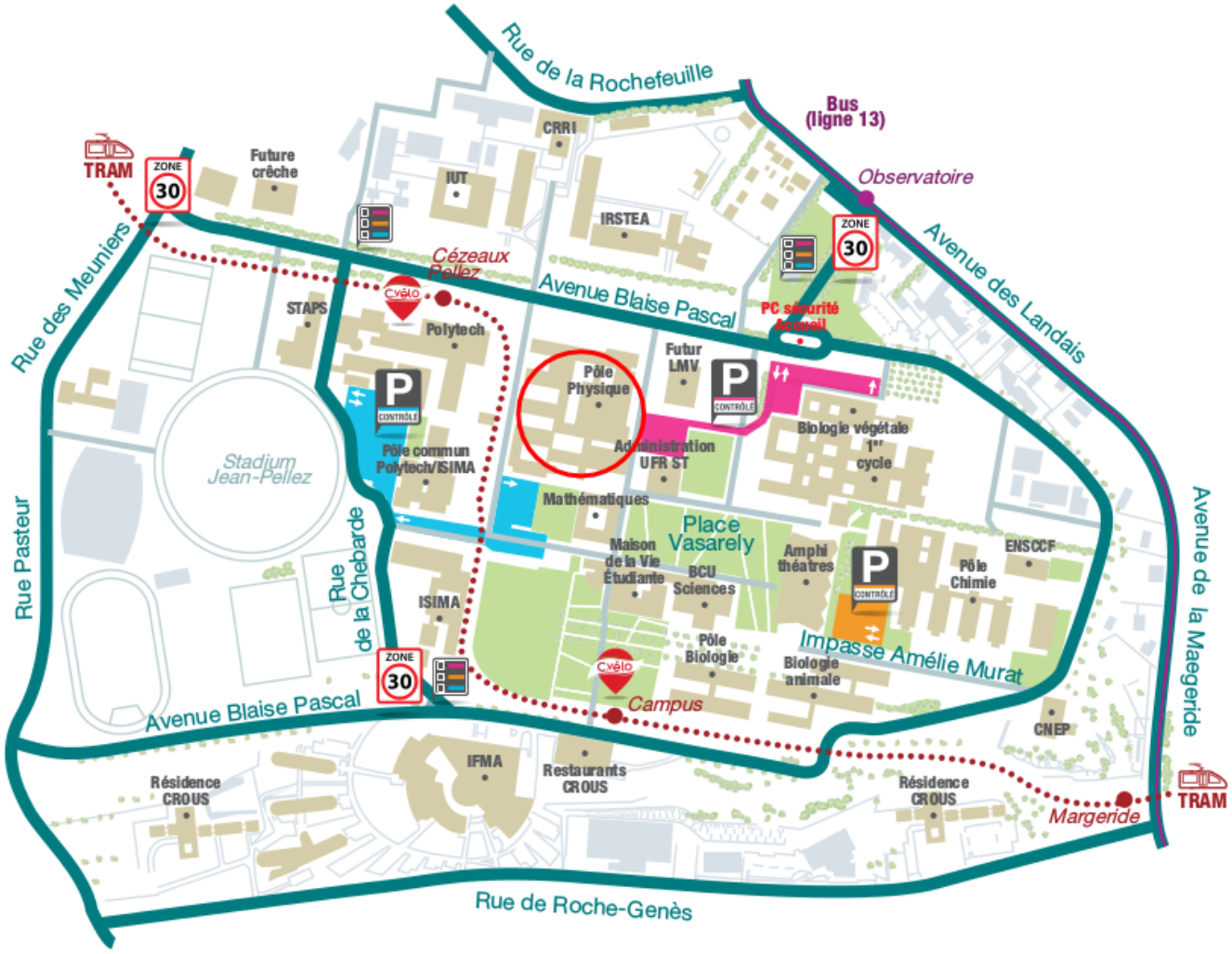

```
\includegraphics[width=.8\linewidth]{pics/landscape.jpg}
```

Vous êtes ici




```
\includegraphics[width=.8\linewidth]{pics/landscape.jpg}
```

Vous êtes ici




```
\includegraphics[width=.8\linewidth]{pics/landscape}
```

Option de contrôle d'insertion d'image

La commande `\includegraphics` accepte une flopée d'option sous la forme `clé=valeur`

Option de contrôle d'insertion d'image

La commande `\includegraphics` accepte une flopée d'option sous la forme `clé=valeur`

`scale` : redimensionner à l'échelle spécifiée (1 → taille originale)

`width` : redimensionner pour avoir la largeur indiquée

`height` : redimensionner pour avoir la hauteur indiquée

`keepaspectratio` : considère les dimensions `width` et `height` comme maximales, et évite de déformer l'image

`rotate` : appliquer une rotation à l'image

`clip` : prendre un rectangle dans l'image

`page` : inclure une page spécifique de l'image (inclusion de `pdf`)

Option de contrôle d'insertion d'image

La commande `\includegraphics` accepte une flopée d'option sous la forme `clé=valeur`

`scale` : redimensionner à l'échelle spécifiée (1 → taille originale)

`width` : redimensionner pour avoir la largeur indiquée

`height` : redimensionner pour avoir la hauteur indiquée

`keepaspectratio` : considère les dimensions `width` et `height` comme maximales, et évite de déformer l'image

`rotate` : appliquer une rotation à l'image

`clip` : prendre un rectangle dans l'image

`page` : inclure une page spécifique de l'image (inclusion de `pdf`)

```
\includegraphics[page=3,rotate=90,scale=.5]{doc}
```

Définir ses propres commandes

Définir une commande

Il est possible de définir des commandes, *via* :

`\newcommand` définit une nouvelle commande

`\renewcommand` redéfinit une commande existante

`\providecommand` définit une commande si elle n'existe pas

Définir une commande

Il est possible de définir des commandes, *via* :

`\newcommand` définit une nouvelle commande
`\renewcommand` redéfinit une commande existante
`\providecommand` définit une commande si elle n'existe pas

Les trois ont la même forme (prenons `\newcommand` pour la suite).

Commandes simples

Une commande simple ne prend pas d'argument.

Commandes simples

Une commande simple ne prend pas d'argument.

Pour la définir, il faut donner

1. son nom (qui commence par '\')
2. son code (le code à exécuter)

Commandes simples

Une commande simple ne prend pas d'argument.

Pour la définir, il faut donner

1. son nom (qui commence par '\')
2. son code (le code à exécuter)

```
\newcommand{\cursus}{L1 info}
Je suis en \cursus.
```

Je suis en L1 info.

Commandes simples

Une commande simple ne prend pas d'argument.

Pour la définir, il faut donner

1. son nom (qui commence par '\')
2. son code (le code à exécuter)

```
\newcommand{\cursus}{L1 info}
Je suis en \cursus.
```

Je suis en L1 info.

```
\newcommand{\cursus}{L1 info}
Je suis en \cursus et pas toi.
```

Je suis en L1 infoet pas toi.

Commandes simples

Une commande simple ne prend pas d'argument.

Pour la définir, il faut donner

1. son nom (qui commence par '\')
2. son code (le code à exécuter)

```
\newcommand{\cursus}{L1 info}
Je suis en \cursus.
```

Je suis en L1 info.

```
\newcommand{\cursus}{L1 info}
Je suis en \cursus\ et pas toi.
```

Je suis en L1 info et pas toi.

Commandes simples

Une commande simple ne prend pas d'argument.

Pour la définir, il faut donner

1. son nom (qui commence par '\')
2. son code (le code à exécuter)

```
\newcommand{\cursus}{L1 info }
Je suis en \cursus.
```

Je suis en L1 info .

```
\newcommand{\cursus}{L1 info }
Je suis en \cursus et pas toi.
```

Je suis en L1 info et pas toi.

Commandes simples

Une commande simple ne prend pas d'argument.

Pour la définir, il faut donner

1. son nom (qui commence par '\')
2. son code (le code à exécuter)

```
\newcommand{\cursus}{L1 info\xspace}
Je suis en \cursus.
```

Je suis en L1 info.

```
\newcommand{\cursus}{L1 info\xspace}
Je suis en \cursus et pas toi.
```

Je suis en L1 info et pas toi.

Le paquet `xspace` définit la commande `\xspace`
qui met un espace quand il y a besoin.

Commandes avec arguments

Les commandes peuvent prendre des arguments.

Commandes avec arguments

Les commandes peuvent prendre des arguments.

Pour définir de telles commandes,

- on indique le nombre d'argument à la commande `\newcommand`
- on utilise leur valeurs, *via* #1 (1^{er} arg), #2 (2^{e} arg), ...

Commandes avec arguments

Les commandes peuvent prendre des arguments.

Pour définir de telles commandes,

- on indique le nombre d'argument à la commande `\newcommand`
- on utilise leur valeurs, *via* #1 (1er arg), #2 (2e arg), ...

```
\newcommand{\strong}[1]{\textbf{#1}\xspace}
La commande \strong{strong} à l'acte.
```

La commande **strong** à l'acte.

Commandes avec arguments

Les commandes peuvent prendre des arguments.

Pour définir de telles commandes,

- on indique le nombre d'argument à la commande `\newcommand`
- on utilise leur valeurs, *via* #1 (1er arg), #2 (2e arg), ...

```
\newcommand{\strong}[1]{\textbf{#1}\xspace}
La commande \strong{strong} à l'acte.
```

La commande **strong** à l'acte.

```
\newcommand{\swap}[2]{#2\xspace#1\xspace}
Et \swap{versa}{vice}.
```

Et vice versa.

Commandes avec un argument optionnel

Après la spécification du nombre d'arguments,
on peut passer un second argument optionnel à `\newcommand` :
la valeur par défaut du premier argument qui devient optionnel

Commandes avec un argument optionnel

Après la spécification du nombre d'arguments,
on peut passer un second argument optionnel à `\newcommand` :
la valeur par défaut du premier argument qui devient optionnel

```
\newcommand{\pluriel}[2][s]{#2#1\xspace}
2 \pluriel{dé}, 1000 \pluriel[x]{jeu}.
```

2 dés, 1000 jeux.

Aller plus loin sur la définition de commande

La commande `\newenvironnement`

permet de définir des environnements.

Aller plus loin sur la définition de commande

La commande `\newenvironnement`
permet de définir des environnements.

Le paquet `xparse` permet de définir des commandes
avec des spécifications avancées d'arguments.

Aller plus loin sur la définition de commande

La commande `\newenvironnement`
permet de définir des environnements.

Le paquet `xparse` permet de définir des commandes
avec des spécifications avancées d'arguments.

Le paquet `xifthen` permet d'écrire des conditions.

Aller plus loin sur la définition de commande

La commande `\newenvironnement` permet de définir des environnements.

Le paquet `xparse` permet de définir des commandes avec des spécifications avancées d'arguments.

Le paquet `xifthen` permet d'écrire des conditions.

Les paquets `keyval` et `pgfkeys` permettent chacun de gérer des listes d'options sous la forme `clé=valeur`

Prochainement

TP : git & LaTeX avec tout ce qu'on a vu

LaTeX : bientôt : l'inclusion de code, les environnements flottants, la gestion des styles

Git : les branches et les fusions

Projet : formez vos binômes

Prochainement

TP : git & LaTeX avec tout ce qu'on a vu

LaTeX : bientôt : l'inclusion de code, les environnements flottants, la gestion des styles

Git : les branches et les fusions

Projet : formez vos binômes

Votre mission

- être à l'aise avec les TP précédents
- vérifier que vous avez accès au gitlab de l'ISIMA

https://gitlab.isima.fr

Prochainement

TP : git & LaTeX avec tout ce qu'on a vu

LaTeX : bientôt : l'inclusion de code, les environnements flottants, la gestion des styles

Git : les branches et les fusions

Projet : formez vos binômes

Votre mission

- être à l'aise avec les TP précédents
- vérifier que vous avez accès au gitlab de l'ISIMA

https://gitlab.isima.fr

Défi

- Comment définir une commande, sans se soucier de si elle existe déjà ou pas ? (voir slide 31)