

# GÉRER SON CODE SOURCE AVEC GIT

DÉCOUVERTE ET BONNES PRATIQUES

ROMAIN NOËL  
PARIS – 17/02/2020



**BNP PARIBAS**

La banque d'un monde qui change

# Déroulé de la formation

## Formation



De 9h00 à 17h30  
1h de break  
Quelques pauses

## Règles



Echange  
Ecoute  
Aide  
Bienveillance

## Participants



Qui êtes-vous et d'où venez-vous ?

## Attentes



Qu'attendez-vous de cette formation ?



**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 2

BNPP Classification : Internal

# SOMMAIRE

## 01 INTRODUCTION

Découverte et premiers pas

## 02 DÉBUTER AVEC GIT

Les commandes de survie

## 03 LES BRANCHES

Support de développement non-linéaire

## 04 FAQ ET COMMANDES AVANCÉES

Pour utilisateurs avertis



**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 3

BNPP Classification : Internal

# **01 - INTRODUCTION**

---

Découverte et premier pas



**BNP PARIBAS**

**La banque d'un monde qui change**

Formation Git | 06/03/2020 | 4

BNPP Classification : Internal

# Exercice pratique : Configurer son poste



**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 5

BNPP Classification : Internal

# Qu'est-ce que git ?

« Git est un logiciel de gestion de versions de sources (décentralisé) »



**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 6

BNPP Classification : Internal

# Qu'est-ce que git ?



## DISPONIBILITÉ ET SÉCURITÉ

Si Jean-Claude n'est pas là, impossible d'intégrer du contenu. De plus, si le PC de Jean-Claude tombe en panne ou se fait voler, une (potentiellement grosse) partie du travail est perdu



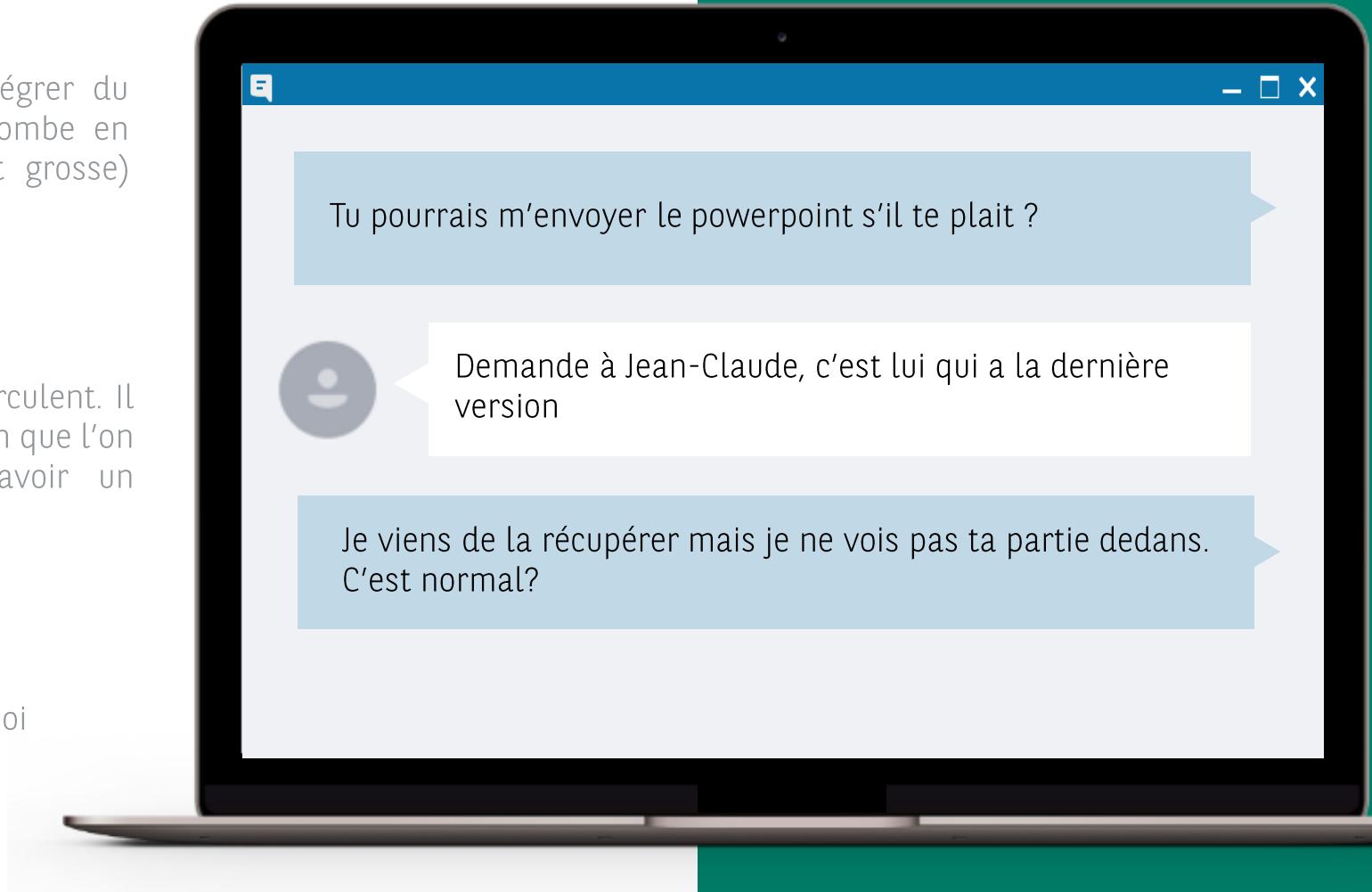
## VERSIONNING

Il existe plusieurs versions du document qui circulent. Il est difficile de savoir quelle est la bonne version que l'on doit utiliser. Il est également difficile d'avoir un historique clair des différentes versions.



## PAS DE TRACABILITÉ

On ne sait pas qui a modifié le fichier, ni pourquoi



**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 7

BNPP Classification : Internal

# Qu'est-ce que git ?

Toute modification est tracée avec un message expliquant sa raison. Il est facile de revenir à un état précédent.

## CENTRALISATION



## HISTORIQUE



Le code est centralisé et sécurisé sur un serveur. Chaque acteur travaille avec une copie locale et se synchronise avec le serveur au fur et à mesure de ses modifications.

## GESTION DE CONFLITS



Grâce à la centralisation, il peut arriver que les travaux de plusieurs personnes se chevauchent. L'outil va alors aider les parties prenantes à gérer ce conflit.

Git permet de fixer le code à n'importe quel moment afin d'en faire une version. Cette version est alors archivée et consultable.

## NON-LINÉAIRE



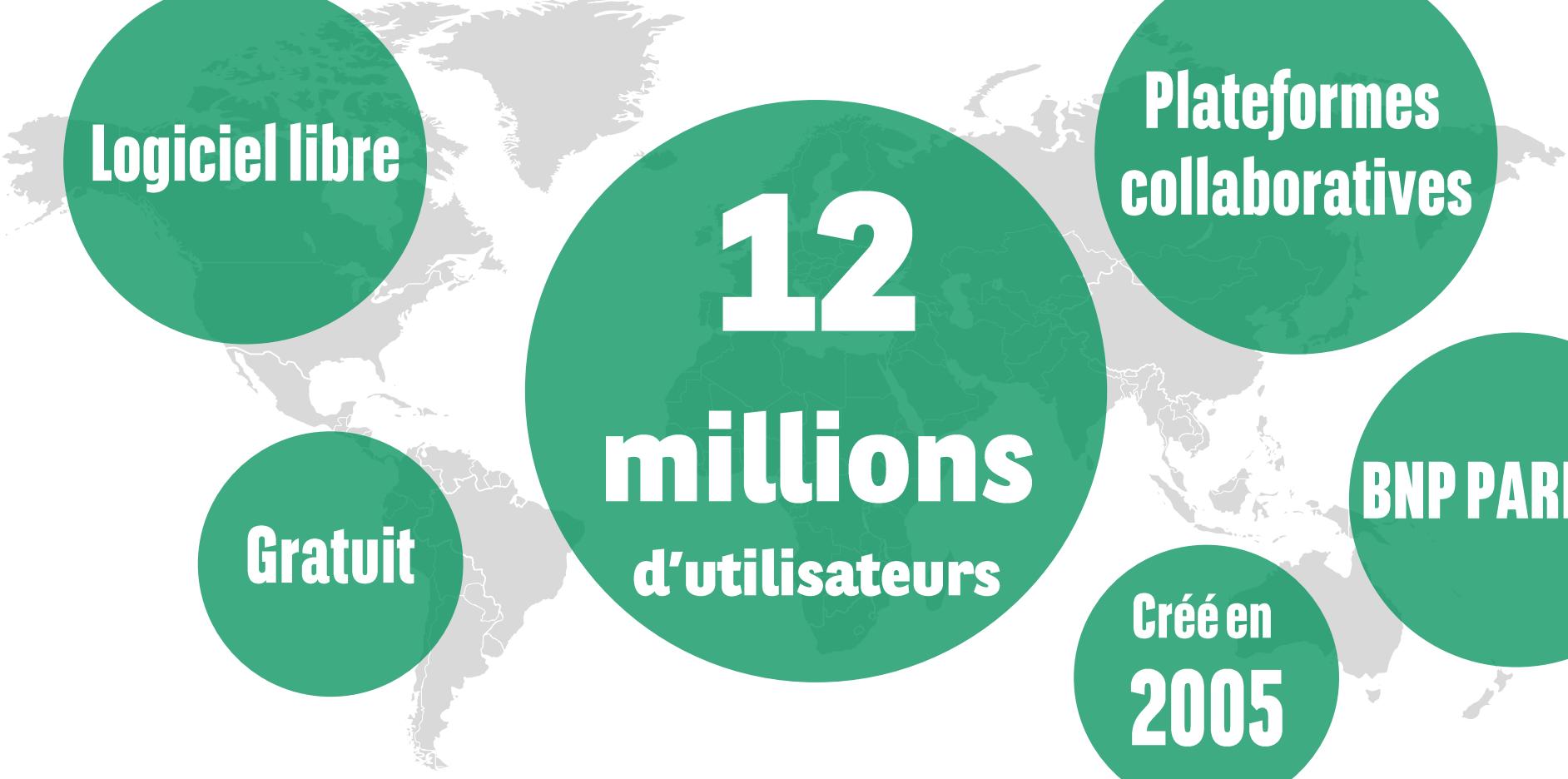
## VERSIONNING



Git offre également un support de développement non-linéaire. Il est possible de partir de n'importe quelle version du code pour faire des expérimentations avant de décider (ou non) de les inclure.



# Pourquoi git ?



Logiciel libre

Gratuit

12 millions  
d'utilisateurs

Plateformes  
collaboratives

BNP PARIBAS

Créé en  
2005



**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 9

BNPP Classification : Internal

# Pourquoi git ?

**«Je ne suis qu'un sale égocentrique, donc j'appelle tous mes projets d'après ma propre personne:D'abord Linux, puis Git.»**

**Linus TORVALDS**



**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 10

BNPP Classification : Internal

# Toolchain (partie 1/2)

## Développement

Etape de production de code. Le code est écrit, puis testé localement sur la machine du développeur. Les principaux outils de développement sont les IDEs parmi lesquels on citera les plus connus :

- Eclipse
- IntelliJ IDEA
- ...



## Gestionnaire de sources

Etape de sauvegarde, de mise en commun et de versionning. Chaque développeur synchronise son travail avec le serveur rassemblant toutes les sources. L'outil le plus communément utilisé est Git.



## Pipeline

### Qualimétrie

Etape de métrologie. Le code est extrait du gestionnaire de sources puis analysé afin de pointer les éventuels défauts.



### Build & Packaging

Lors de cette étape, le code est compilé, puis packagé en une archive prête à être déployée.



## Repository

Sauvegarde de l'archive dans un endroit (repository) sécurisé



Toutes ces étapes sont dirigées par un orchestrateur, configurant ainsi de manière automatique les outils de ces différentes étapes.



**BNP PARIBAS**

**La banque d'un monde qui change**

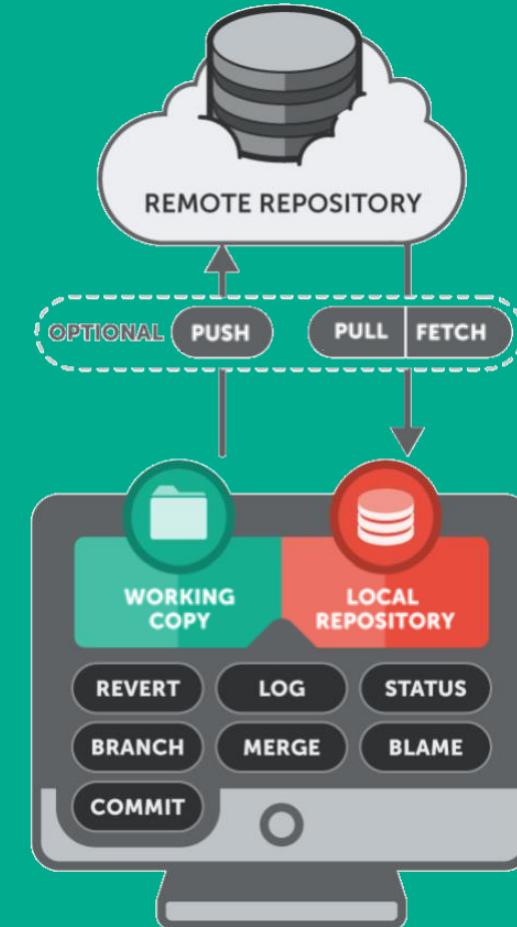
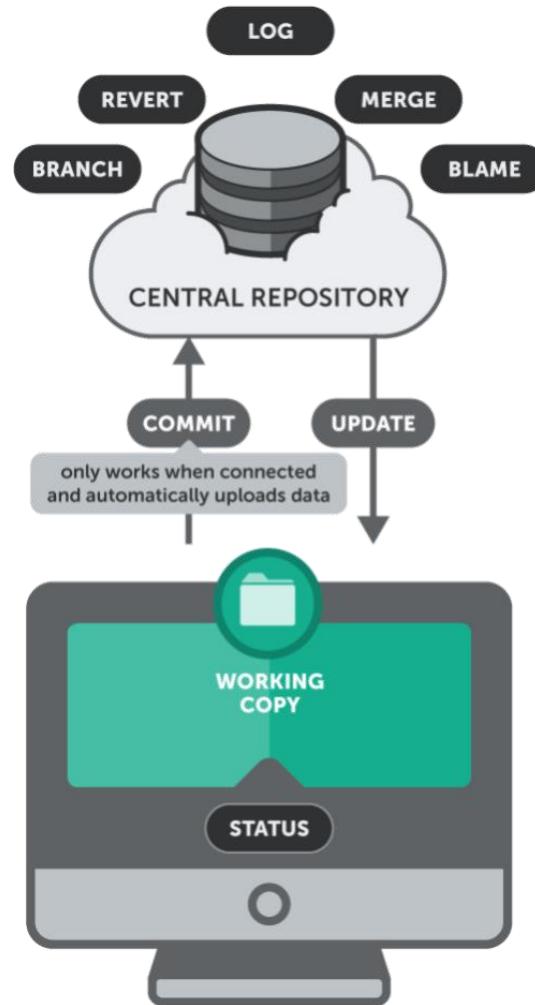
Formation Git | 06/03/2020 | 11

BNPP Classification : Internal

# Toolchain (partie 2/2)



# SVN vs git



# 02 - DÉBUTER AVEC GIT

---

Les commandes de survie



**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 14

BNPP Classification : Internal

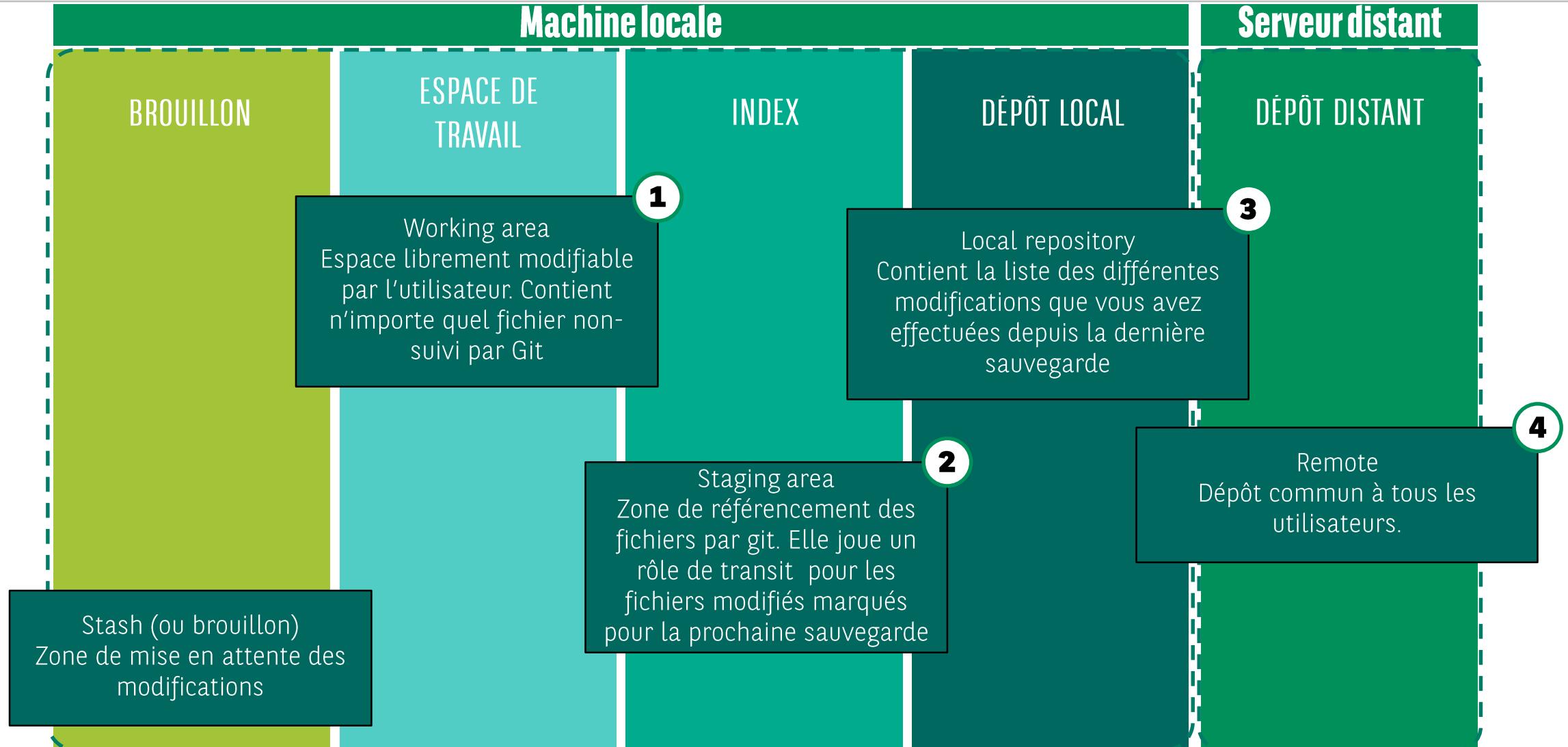
- **Git agit sur vos fichiers comme machine à état**

Git garde en mémoire l'état de chacun de vos fichiers. Ces derniers peuvent-être :

- Nouveaux
- Modifiés mais non-sauvegardés
- À jour avec le serveur
- Etc...

- **Toute commande permettra de faire passer un fichier d'un état à un autre**

# Les différents espaces git



# Indexation git



## INDEXER LE FICHIER

Avant toute sauvegarde, il faut lister les fichiers qui sont à prendre en compte pour la prochaine (et uniquement la prochaine) sauvegarde

```
git add <nom_du_fichier>
```



## DÉ-INDEXER LE FICHIER

En cas d'erreur, on peut supprimer l'indexation du fichier

```
git reset <nom_du_fichier>
```



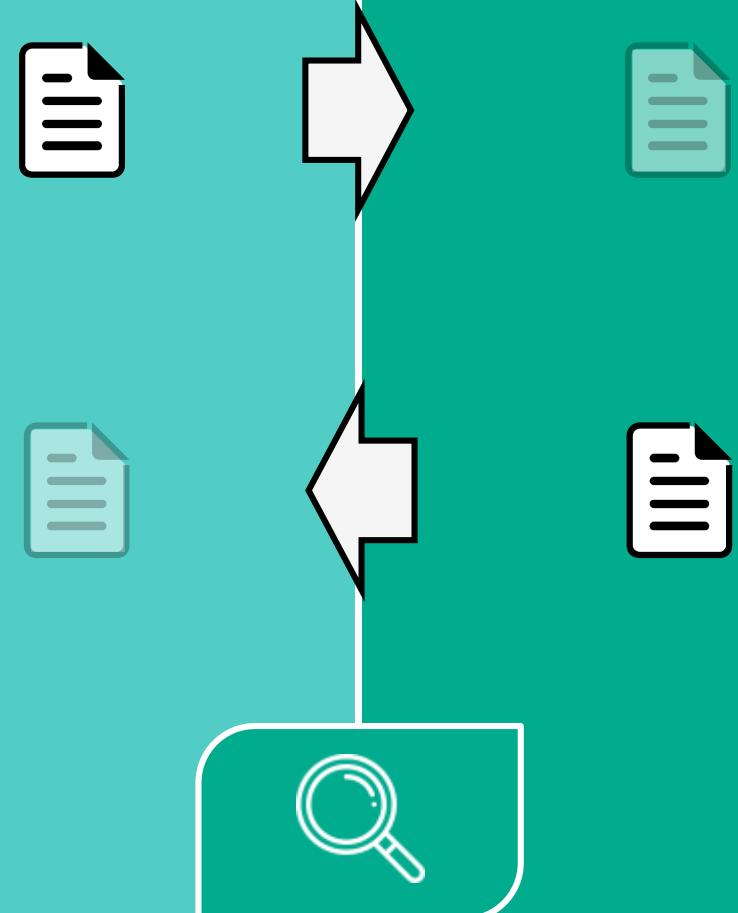
## VÉRIFIER L'INDEX

On peut vérifier l'état de des fichiers à tout moment. Cela permet de savoir si les fichiers sont nouveaux, trackés, modifiés...

```
git status
```

ESPACE DE  
TRAVAIL

INDEX



BNP PARIBAS

La banque d'un monde qui change

Formation Git | 06/03/2020 | 17

BNPP Classification : Internal

# Exercice pratique :

## Commandes add, reset et status



**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 18

BNPP Classification : Internal



## En cas de suppression de fichier

En cas de suppression de fichier, il faut notifier git de ce changement. Cette notification se fait de la même manière qu'une modification grâce à la commande git add.

## .gitignore

.gitignore est un fichier à la racine de votre dossier comprenant une liste de fichiers à ne jamais prendre en compte lors de vos indexations. Cela peut être pratique pour ne jamais partager vos fichiers de configuration d'IDE, vos sources compilées, ou vos ressources contenant des variables secrètes.

```
13 .idea/  
14 .settings/  
15 *.SWO  
16 modules/.settings  
17 .bazelrc  
18 .vscode  
19 modules/.vscode
```



## Les options utiles de la commande

Il est possible d'indexer plusieurs fichiers simultanément.

### 1 - De manière exhaustive

Il est possible de lister de manière exhaustive tous les fichiers à ajouter, séparés par des espaces.

### 2 - La totalité du répertoire courant

Grâce aux commandes "git add ." ou "git add -A", git indexe la totalité des fichiers dans le répertoire courant (récursevement dans les dossiers)

### 3 – En reprenant la liste de la dernière sauvegarde

La commandes "git add -u" permet à git d'ajouter la liste des fichiers déjà indéxes lors de la dernière sauvegarde.



# Les sauvegardes git



## CRÉER UNE SAUVEGARDE

Maintenant que vous avez listé les fichiers dont vous voulez faire une sauvegarde, il est temps de créer une sauvegarde

```
git commit -m "message"
```



## ANNULER LA DERNIÈRE SAUVEGARDE

En cas d'erreur dans un commit, on peut annuler le dernier commit

```
git reset HEAD^
```



## VÉRIFIER LA LISTE DES COMMITS

Pour lister toutes les sauvegardes, leurs auteurs et leurs messages associés.

```
git log
```

INDEX

DÉPÔT LOCAL



BNP PARIBAS

La banque d'un monde qui change

Formation Git | 06/03/2020 | 20

BNPP Classification : Internal

# Exercice pratique :

## Commandes commit, reset et log



**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 21

BNPP Classification : Internal

# Git commit : Bonnes pratiques



## Un bon message de commit, pourquoi est-ce important ?

- Pour comprendre l'historique des modifications.
- Le travail de développement consiste à écrire du code 10% du temps et 90% à en lire.
- Pour faciliter l'analyse a posteriori
- Un bon commentaire est un gain de performances.
- Il est un indicateur fondamental pour l'activité de Release Management.



L'enjeu est de donner le plus de contexte possible à chaque modification. Elle doit être compréhensible rapidement par n'importe qui.



## Quelques conseils et bonnes pratiques

- Faire des petits commits
- Limiter le nombre de caractères pour faire en sorte que le sujet soit explicite.
- Toujours expliquer le « pourquoi » et rarement le « comment »
- Contient l'identifiant du ticket JIRA.
- Répond à un format particulier (il est possible de forcer l'utilisation d'un template)

COMMENT	DATE
O CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
O ENABLED CONFIG FILE PARSING	9 HOURS AGO
O MISC BUGFIXES	5 HOURS AGO
O CODE ADDITIONS/EDITS	4 HOURS AGO
O MORE CODE	4 HOURS AGO
O HERE HAVE CODE	4 HOURS AGO
O AAAAAAAA	3 HOURS AGO
O ADKFJSLKDFJSOKLFJ	3 HOURS AGO
O MY HANDS ARE TYPING WORDS	2 HOURS AGO
O HAAAAAAAAANDS	2 HOURS AGO





```
git commit --amend -m "nouveau message"
```

## MODIFIER LE DERNIER COMMIT

- **Pour conserver un historique propre**

La principale raison d'effectuer un ammend est d'éviter de créer plusieurs commits pour une seule responsabilité. De cette manière, vous évitez de créer des sauvegardes dont la granularité trop fine empêcherait d'avoir une vision claire de l'historique des changements.

- **Pour réécrire le message de commit**

Si le message que vous avez entré lors de votre dernier commit ne vous convient pas, ou tout simplement si vous vous êtes trompés, la commande « git commit --ammend » vous permet de réécrire ce message. Si toutefois vous ne souhaitez pas réécrire de message et uniquement ajouter ou supprimer des fichiers à indexer, vous pouvez utiliser l'option « --no-edit ».



# Publier son code



## PUBLIER DES MODIFICATIONS

Dernière étape permettant de rendre disponible les modifications incrémentales au reste du monde

```
git push
```



## ANNULER UNE SAUVEGARDE PUBLIÉE

Une fois un commit publié, il est difficile de l'annuler. La solution la plus simple est de créer un commit inverse, puis de publier cet inverse pour annuler la modification souhaitée.

```
git revert #commit
```



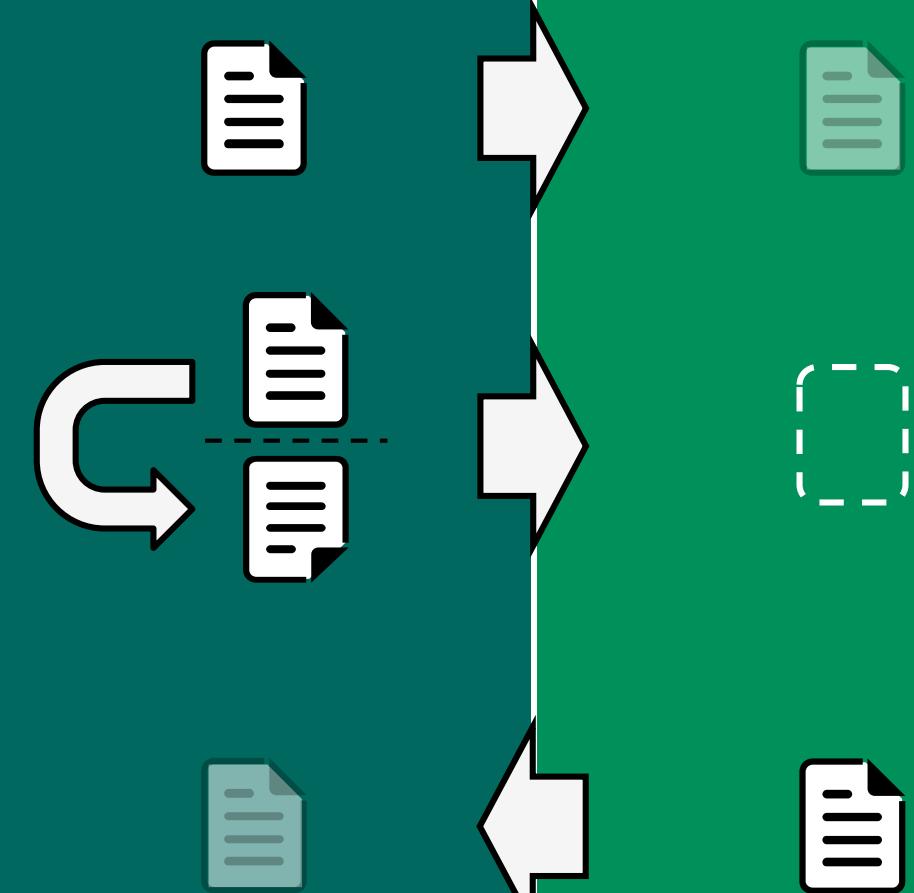
## RÉCUPÉRER DES MODIFICATIONS

Cette commande récupère les commits du serveur distant, puis fusionne les changements avec votre dépôt local.

```
git pull
```

DÉPÔT LOCAL

DÉPÔT DISTANT



**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 25

BNPP Classification : Internal

# Exercice pratique (en binôme) : Commandes push, pull et revert



**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 26

BNPP Classification : Internal

# Gérer les conflits

## • Impossibilité de faire un push

Si vous essayez de publier des modifications alors que votre dépôt local n'est pas à jour, git vous en empêchera et vous demandera de vous mettre à jour avant de continuer.

## • Conflits de fusion

Dans le cas où vos modifications se chevaucheraient avec celles d'un autre développeur (e.g : modification de la même ligne d'un fichier), git vous demandera de gérer le conflit avant de créer un nouveau commit.

## • Conflits sur une zone de code

La zone précise entrant en conflit est notifiée par git par une suite de symboles. La première zone (HEAD) est celle de votre poste, la seconde (#commit) est celle du serveur distant. A vous d'effectuer la fusion des codes, de créer un nouveau commit et de les publier.

```
$ git push
To https://collaborate.mobypic.dev.echonet/os-sit-ap00000/crafters/tppgit.git
 ! [rejected]      master -> master (fetch first)
error: failed to push some refs to 'https://collaborate.mobypic.dev.echonet/os-sit'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

```
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://collaborate.mobypic.dev.echonet/os-sit-ap00000/crafters/tppgit
 7b52b5b..89fc9f2  master      -> origin/master
Auto-merging test
CONFLICT (add/add): Merge conflict in test
Automatic merge failed; fix conflicts and then commit the result.
```

```
<<<<< HEAD
System.out.println("Bonjour, moi c'est Romain !");
=====
System.out.println("Hello");
>>>>> ab595f114d10c9d7050243288fff4d94b361bc2b
```



# Les bonnes pratiques du push



Lors de la création d'un commit, il est préférable de regrouper des changements qui vont ensemble. De petits commits unitaires favorisent des publications plus claires et plus fréquentes.

**PETIT**



Réécrire l'historique distant est une très mauvaise pratique! Une fois publié, considérez un commit pushé comme étant irréversible.

**IRREVERSIBLE**



Un commit ne devrait jamais "casser le build". Veillez toujours à vérifier que vos modifications fonctionnent et ne touchent pas à l'intégrité du code.

**FIABLE**



Une fréquence élevée vous forcera à fractionner vos grosses tâches en développement plus petites et évitera l'effet "big-bang".

**FREQUENT**



Un des buts premiers de git est de conserver un historique clair. L'idée est donc de s'assurer d'un dépôt local propre (commits clairs et parlants) avant d'effectuer un push.

**CLAIR**



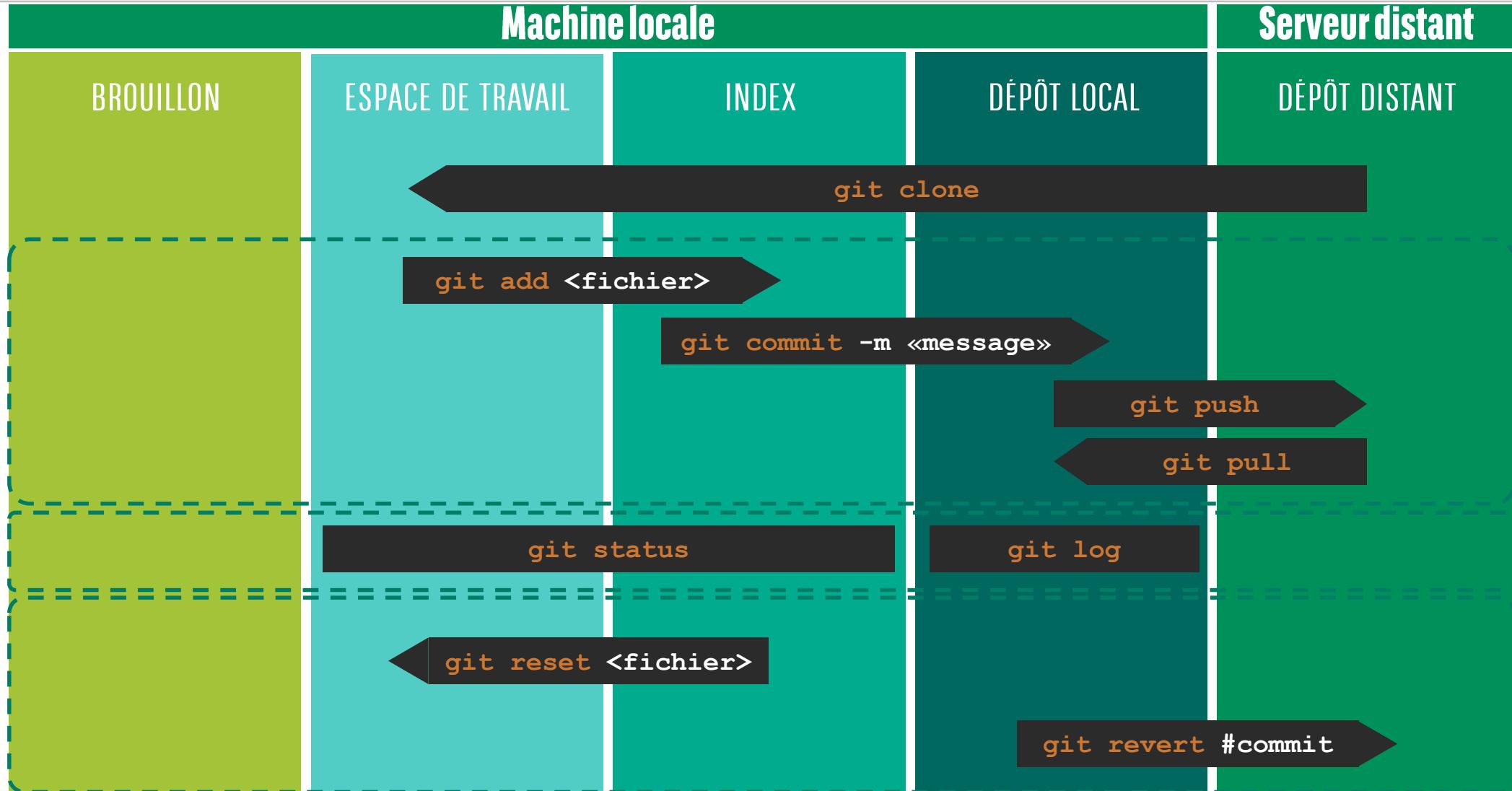
**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 28

BNPP Classification : Internal

# Pour résumer



**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 29

BNPP Classification : Internal

# 03 - COMMITS ET BRANCHES

---

Un support de développement non-linéaire



**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 30

BNPP Classification : Internal

**«Après quelques expérimentations,  
je souhaiterais revenir à un état précis de mon  
historique.»**

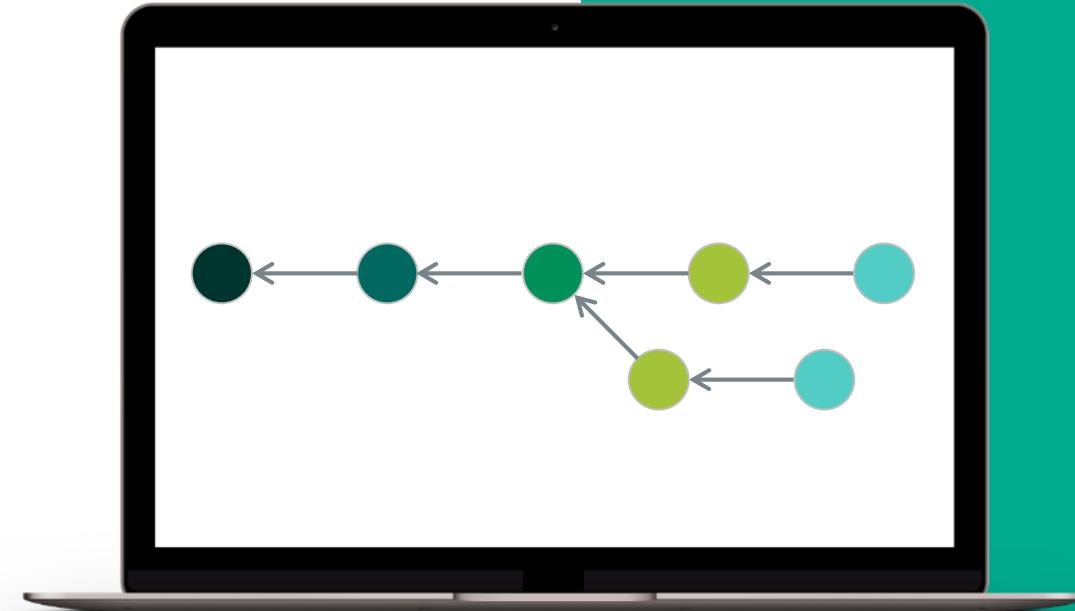
Il est tout à fait possible de se déplacer dans l'historique de son code grâce à Git, c'est même une de ses fonctionnalités principales.



# STRUCTURE D'ÉLÉMENT: LA LISTE CHAÎNÉE

Une liste chaînée désigne en informatique une structure de données représentant une collection ordonnée et de taille arbitraire d'éléments de même type, dont la représentation en mémoire de l'ordinateur est une succession de cellules faites d'un contenu et d'un pointeur vers une autre cellule.

Wikipédia

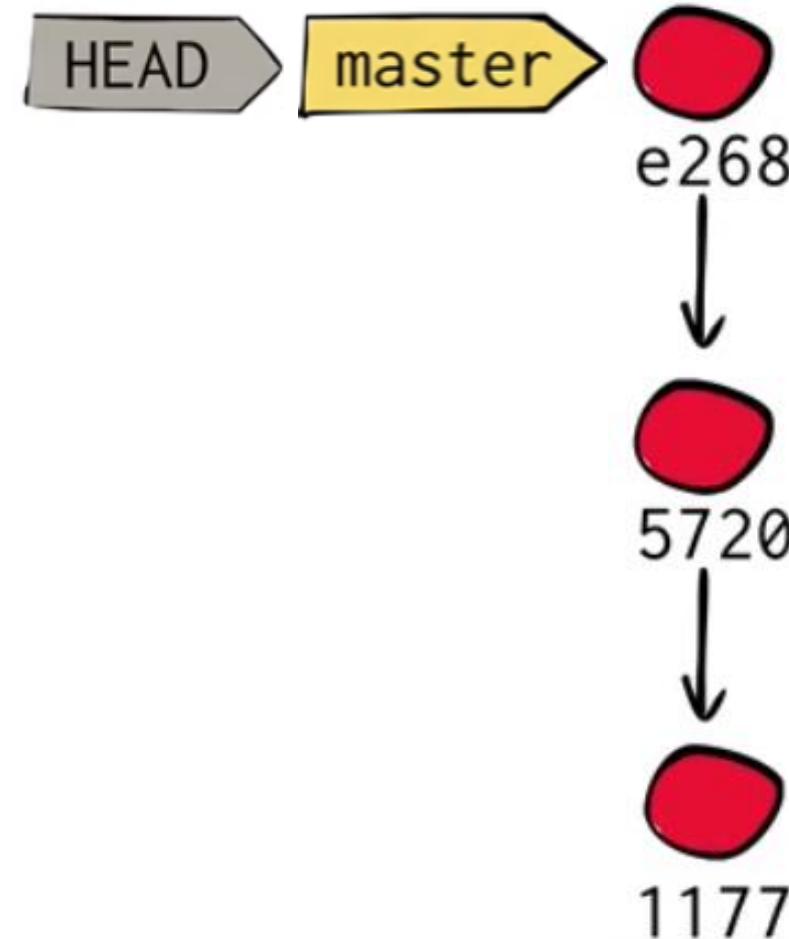


Chaque commit est défini par un **identifiant** (ou hash), un **contenu** (liste des modifications effectuées), et l'**identifiant du commit précédent**. Chaque maillon de cette chaîne représente donc une photographie du file-system à un instant T.

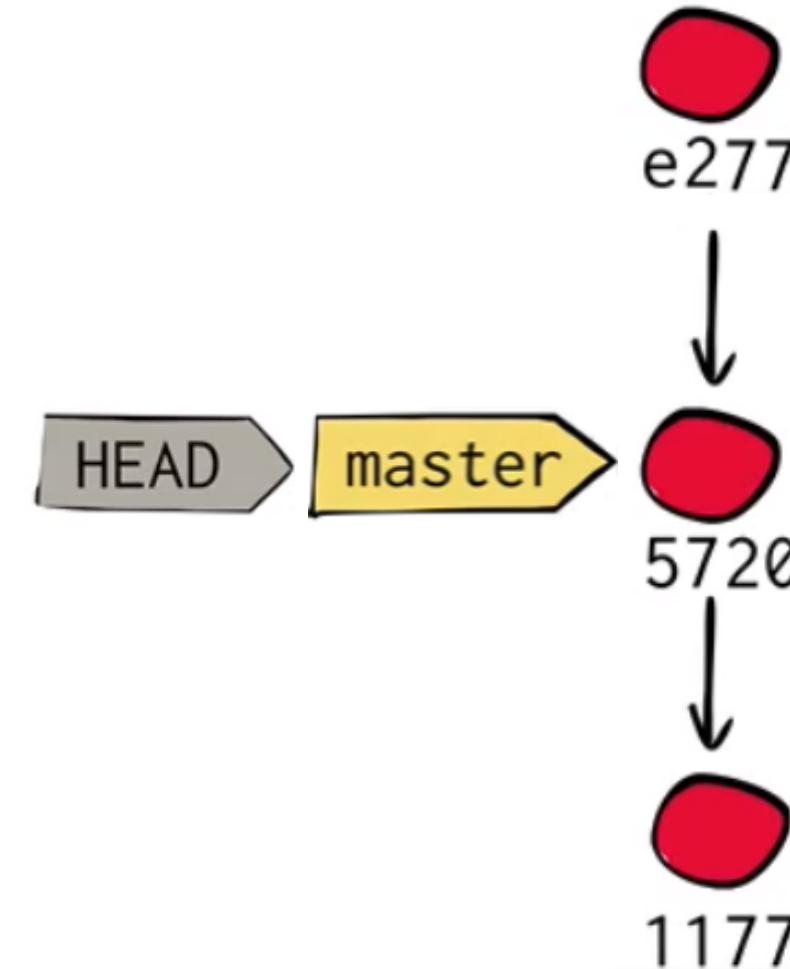
Votre pointeur “courant” s’appelle **HEAD**. Se déplacer dans l’historique du code revient donc à déplacer la HEAD de commits en commits.

## git checkout [commit]

Les identifiants de commits sont complexes et difficiles à mémoriser. Grâce aux mécanismes de **tags** ou de **branches**, il est possible de créer et de nommer des pointeurs vers différents commits.



```
$ git clone https://github.com/BNP-PARIBAS/BNP-Paribas-Website.git  
$ touch nouveau_fichier  
$ git add nouveau_fichier  
$ git commit -m "ajout nouveau fichier"  
  
$ git checkout 1177  
  
$ git checkout master  
  
$ git checkout 5720
```



# Branches vs Tags



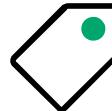
## Les branches : Un support de développement non-linéaire

Créer une branche signifie diverger de la ligne principale de développement et continuer à travailler sans impacter cette ligne. Une fois le travail terminé, cette branche sera fusionnée avec la branche principale (ou pas).

### Exemple

Corinne travaille sur une grosse fonctionnalité. Pour éviter de perturber le travail du reste de l'équipe, elle se crée une branche qu'elle fusionnera pour l'intégrer à la version en cours.

```
git checkout -b <nomBranche>
git branch -a
git branch -d <nomBranche>
```



## Les tags : Un moyen de marquer une version fixe du code

Créer un tag revient à marquer une version du code, un point spécifique dans l'historique. Un tag n'est pas fait pour évoluer et ne devrait pas changer de commit sur lequel il pointe.

### Exemple

Corinne déploie une version de son application en production. Elle crée un tag "prod-v1.0" sur le dernier commit. De cette manière, elle peut facilement retrouver le code déployé afin d'investiguer un éventuel bug.

```
git tag <nomTag>
git tag --list
git tag -d <nomTag>
```



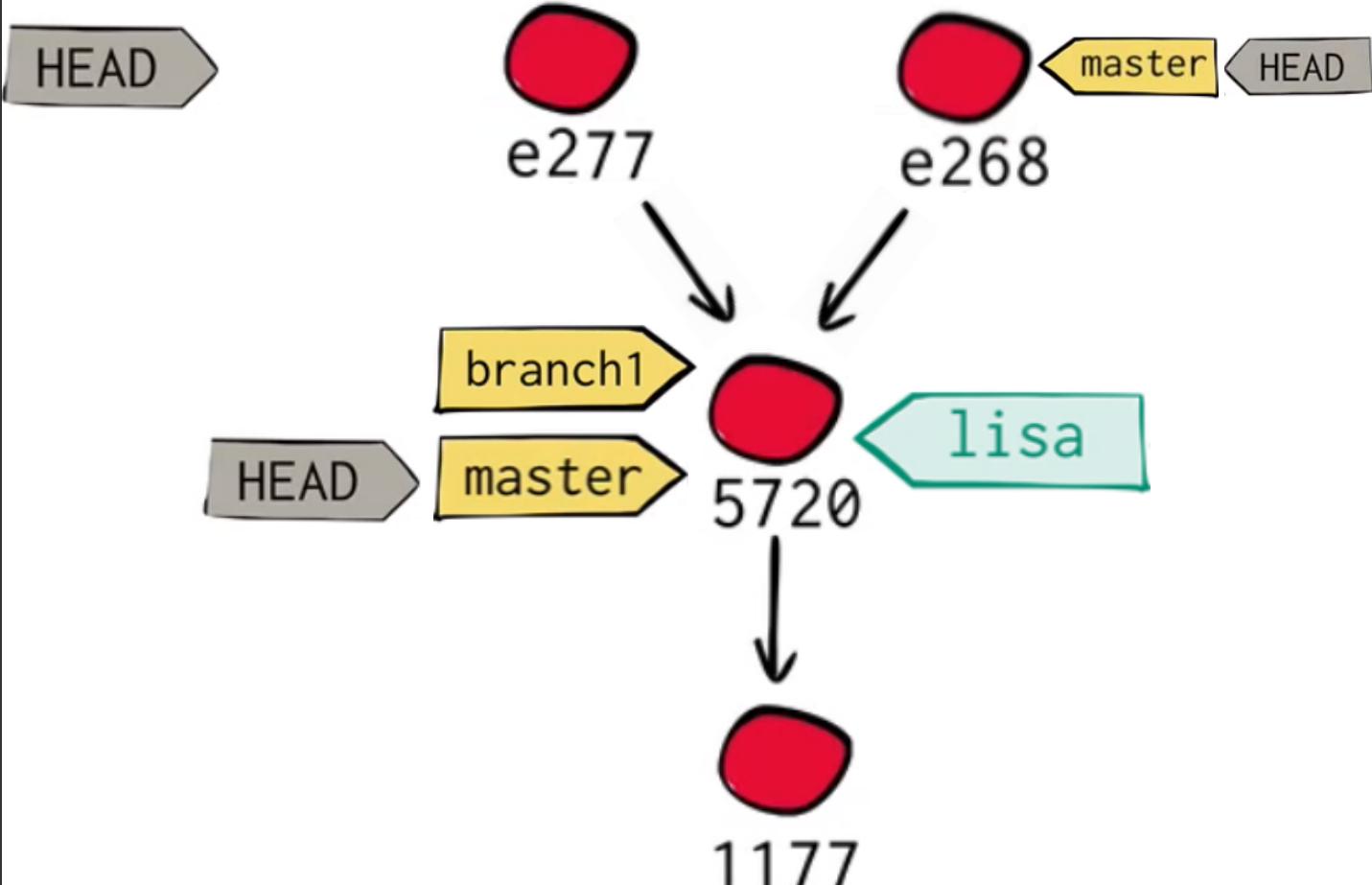
**BNP PARIBAS**

La banque d'un monde qui change

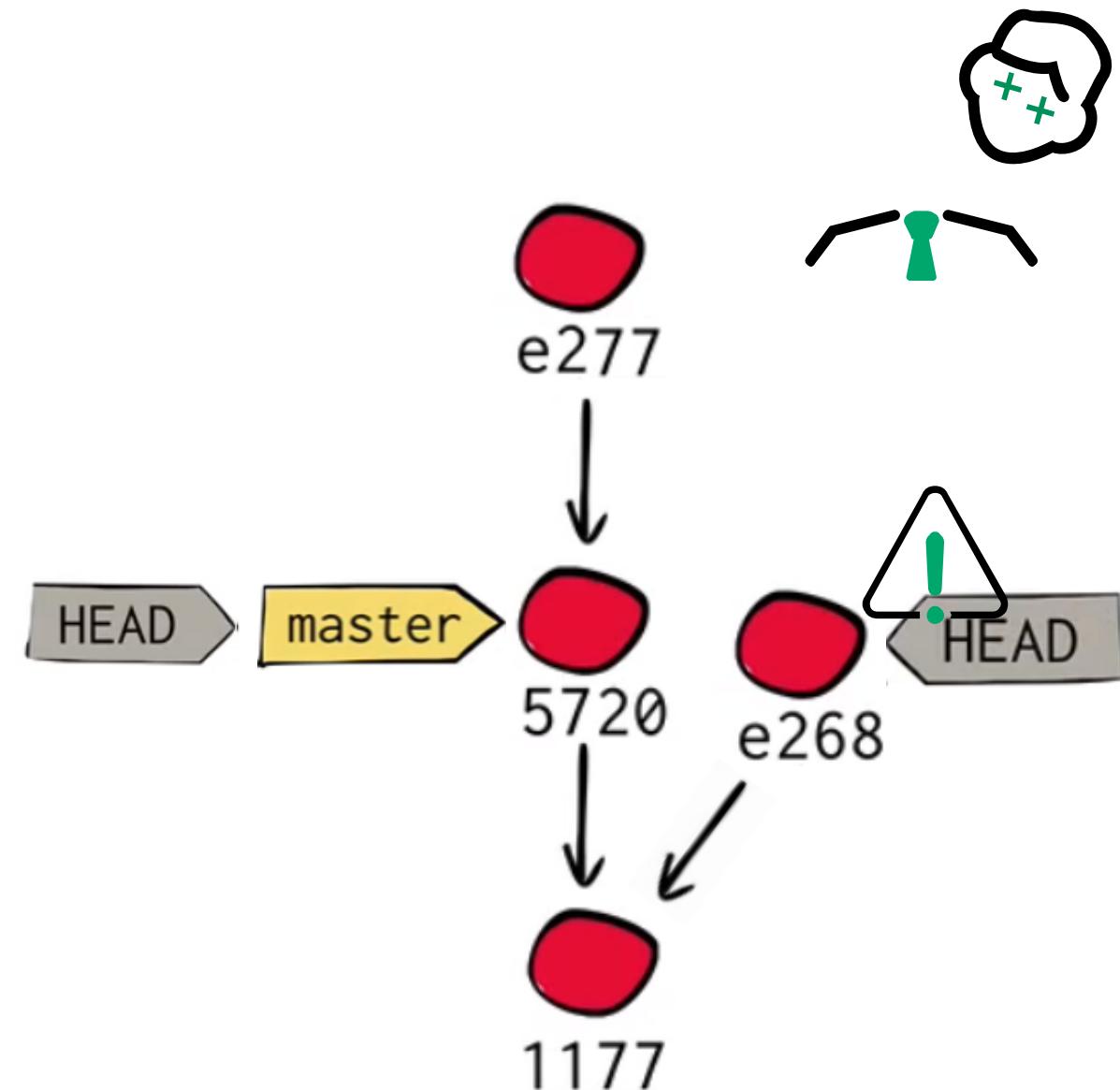
Formation Git | 06/03/2020 | 35

BNPP Classification : Internal

```
$ git clone  
$ git tag lisa  
$ git checkout -b branch1  
$ touch fichier1  
$ git add fichier1  
$ git commit -m "ajout de fichier1"  
  
$ git checkout master  
$ touch fichier2  
$ git add fichier2  
$ git commit -m "ajout de fichier2"  
  
$ git checkout branch1
```



```
$ git clone  
  
$ git checkout 1177  
You are in 'detached HEAD' state  
  
$ touch fichier1  
$ git add fichier1  
$ git commit -m "ajout de fichier1"  
  
$ git checkout master  
  
$ touch fichier2  
$ git add fichier2  
$ git commit -m "ajout de fichier2"
```



# Exercice pratique :

## Commandes checkout, tag et branch

# Les différents workflows



## UTILISER DES BRANCHES

C'est un des principaux atouts de git qui le fait de manière légère et rapide. Travailler sur des branches permet de travailler à plusieurs sur des tâches différentes sans se marcher sur les pieds !



## UNE STRATÉGIE ADAPTÉE

Il existe plusieurs stratégies de répartition de branches (trunk-based, feature branching, gitflow...). Chaque stratégie a ses qualités et ses défauts, mais l'important est de suivre la stratégie définie



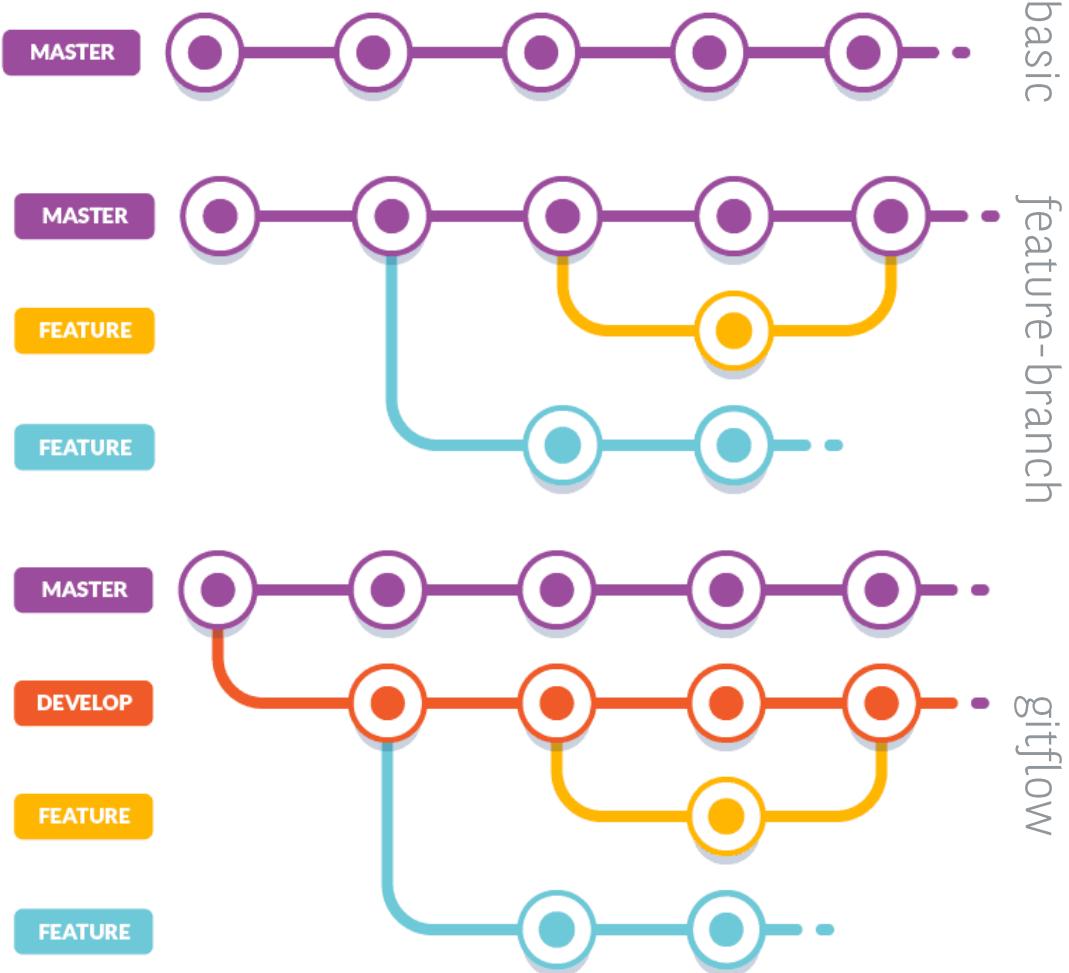
## DURÉE DE VIE COURTE

Cela peut varier en fonction des stratégies, mais en général, il est préférable d'avoir des branches à courte durée de vie (quelques jours).



## UNE BRANCHE = UN DÉVELOPPEUR

Travailler à plusieurs sur une même branche demande beaucoup de rigueur et impose beaucoup de contraintes. En général, cela implique le fait que la story devrait être découpée (story INVEST).

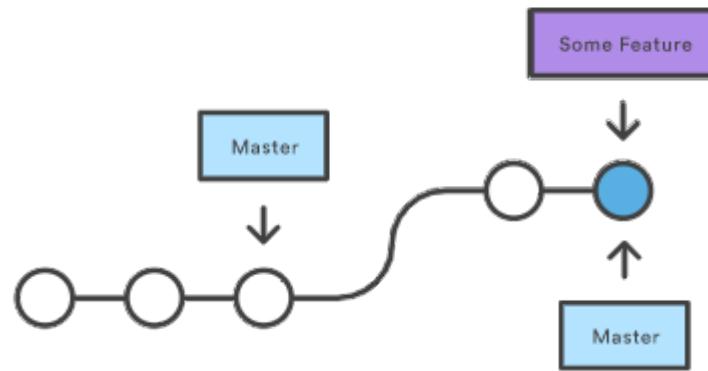


# Les différents types de fusions

`git merge <branche à absorber>`

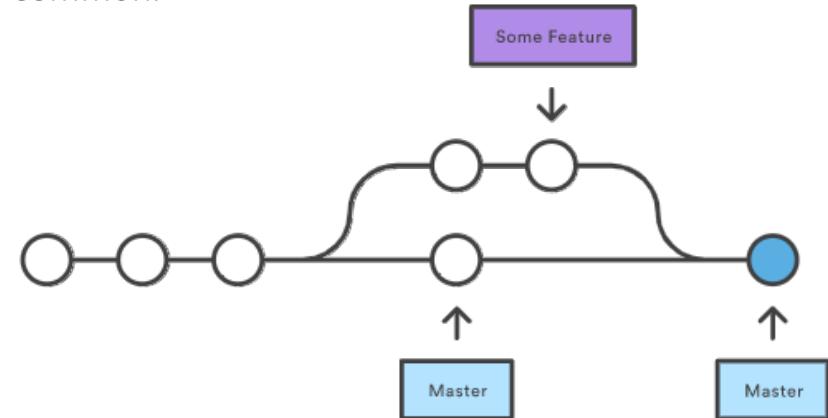
## FAST-FORWARD MERGE

Un fast-forward merge peut se produire lorsqu'il existe un chemin linéaire entre la pointe de branche actuelle et la branche cible. Au lieu de fusionner les branches, tout ce que Git a à faire pour intégrer les historiques est de déplacer le pointeur de branche actuelle jusqu'à la pointe de branche cible.



## 3-WAY MERGE

Cependant, un fast-forward merge n'est pas possible si les branches ont divergé. Lorsqu'il n'y a pas de chemin linéaire entre les deux branches, git est obligé de créer un commit « technique » supplémentaire afin de combiner les différentes versions. La nomenclature vient du fait que Git utilise trois commits pour générer le commit de fusion: les deux extrémités de branche et leur ancêtre commun.

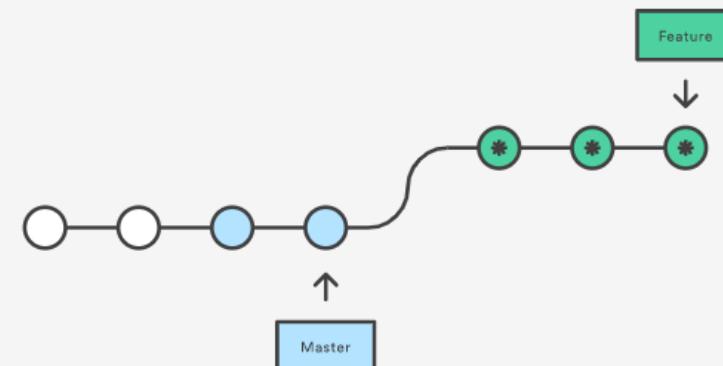
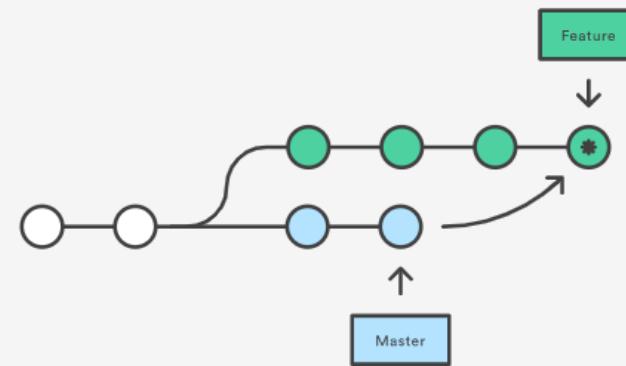
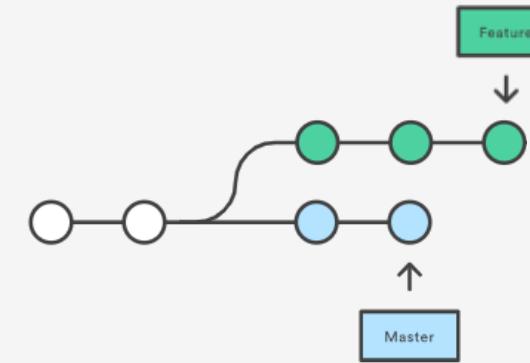


Le rebase est une commande permettant de réécrire l'historique afin de déplacer une branche d'un commit à un autre (en général, la pointe d'une branche).

## git rebase [branche]

Au lieu d'utiliser un commit de fusion, git rejoue l'entièreté des commits de la branche en se basant sur le commit spécifié

Un rebase permet d'éviter les commits « techniques » et transforme tout merge de branche en fast-forward merge



merge

rebase

# Exercice pratique :

## Commandes merge et rebase



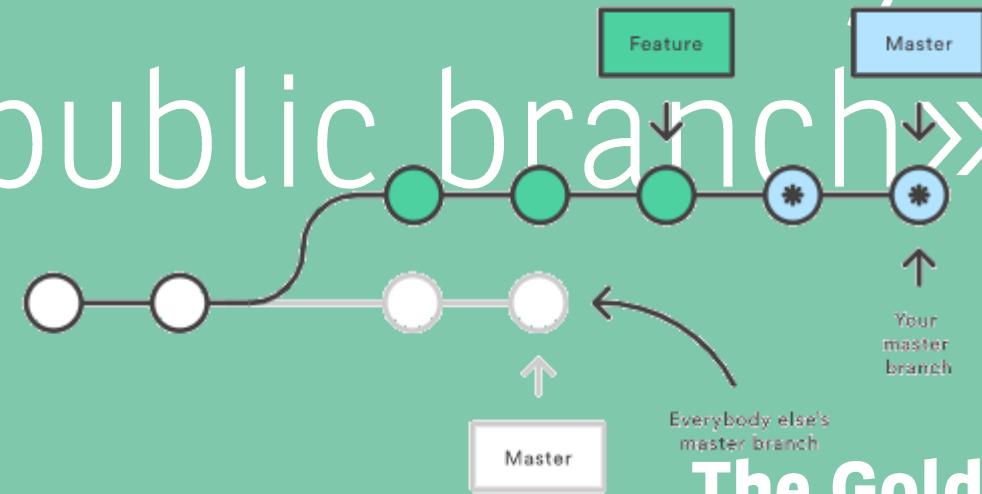
**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 42

BNPP Classification : Internal

# «Never rebase while you're on a public branch»



## The Golden Rule of Rebasing



**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 43

BNPP Classification : Internal

# 04 – FAQ ET COMMANDES AVANCÉES

---

Pour utilisateurs téméraires



**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 44

BNPP Classification : Internal

# Problèmes courants

## IMPOSSIBILITÉ DE FAIRE UN PUSH : GIT NE SAIT PAS SUR QUELLE BRANCHE PUSHER

Lors de vos premiers git push, vous serez confronté à l'erreur suivante. Elle est due au fait que vous n'avez pas spécifié à git sur quelle branche effectuer le push.



## SPÉCIFIER LA BRANCHE À METTRE À JOUR

Soit vous spécifiez à git sur quelle branche effectuer le push, soit vous vous servez de la commande « --set-upstream » pour lui donner une branche par défaut et simplifier votre commande sur les push à venir.

```
$ git push  
fatal: The current branch feature has no upstream branch.  
To push the current branch and set the remote as upstream, use  
  
git push --set-upstream origin feature
```

```
$ git push --set-upstream origin master
```

----- ou -----

```
$ git push origin master
```



**BNP PARIBAS**

La banque d'un monde qui change

Formation Git | 06/03/2020 | 45

BNPP Classification : Internal

# Problèmes courants

## IMPOSSIBILITÉ DE FAIRE UN PUSH: VOTRE DÉPÔT LOCAL N'EST PAS À JOUR

Git ne peut effectuer de fusion que si les différents commits que vous avez effectués se sont faits sur une branche à jour avec le repo. Git vous en notifiera et vous demandera de vous mettre à jour en effectuant un « git pull »



## RÉCUPÉRER LA DERNIÈRE VERSION

Il s'agit ici de mettre son dépôt local à jour. La solution la plus simple est d'effectuer un « git pull » mais vous pouvez tout aussi bien utiliser les commandes fetch+merge ou encore fetch+rebase.

```
$ git push origin master
To http://git-pf.itp.echonet/training/my-project.git
 ! [rejected]      master -> master (non-fast-forward)
error: failed to push some refs to 'http://git-
pf.itp.echonet/training/my-project.git'
hint: Updates were rejected because the tip of your current branch
is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for
details.
```



```
$ git pull
[...]
$ git push origin master
```

# Problèmes courants

## IMPOSSIBILITÉ DE FAIRE UN CHECKOUT: VOUS AVEZ DES MODIFICATIONS EN ATTENTE

Git ne vous laissera pas effectuer de checkout si vous avez des modifications en attente (des modifications non-committées).



## RÉCUPÉRER LA DERNIÈRE VERSION

Il existe deux principales solutions à ce problème. La première est de sauvegarder vos modifications avec un git commit. Cependant, il se peut qu'à cet instant, votre code soit encore instable. Dans ce cas, on préférera donc la commande « git stash ».

```
$ git checkout feature
error: Your local changes to the following files would be
overwritten by checkout:
    somefile.txt
Please commit your changes or stash them before you switch
branches.
Aborting
```

```
$ git commit -A -m "Fixing bug #12345"
$ git checkout feature
```

----- ou -----

```
$ git stash
$ git checkout feature
[...]
$ git checkout yourbranch
$ git stash apply
```



**BNP PARIBAS**

La banque d'un monde qui change

| 06/03/2020 | 47

BNPP Classification : Internal

# Problèmes courants

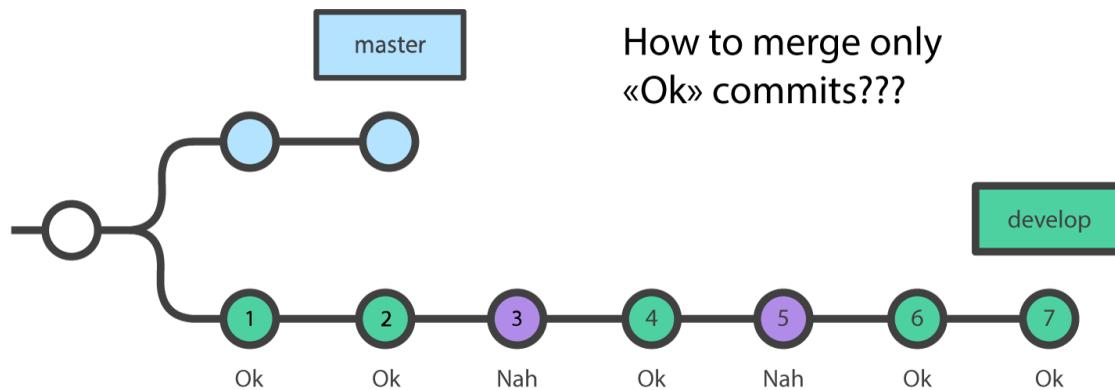
## RÉCUPÉRER UN COMMIT PRÉCIS



Git ne vous laissera pas effectuer de checkout si vous avez des modifications en attente (des modifications non-committées).

## LA COMMANDE CHERRY-PICK

La commande cherry-pick permet de ne sélectionner et de n'appliquer que certains commit bien précis sur la branche courante. On peut spécifier les commits un par un ou donner un intervalle à appliquer.



```
$ git merge commit2  
$ git cherry-pick commit4  
$ git cherry-pick commit5..develop
```



**BNP PARIBAS**

La banque d'un monde qui change

| 06/03/2020 | 48

BNPP Classification : Internal

# Problèmes courants

## SUPPRIMER TOUTES SES MODIFICATIONS POUR SE SYNCHRONISER AVEC LE DÉPÔT DISTANT

Il arrive que les modifications que vous avez effectuées étaient purement exploratoires et ne vous satisfassent pas. Dans ce cas, vous allez vouloir supprimer toutes vos modifications et le remettre dans un état similaire avec le dépôt distant.



## GIT RESET --HARD



Deux solutions radicales. La première consiste à supprimer son dossier courant et à en télécharger une nouvelle copie. La seconde consiste à utiliser la commande « git reset --hard » qui va réinitialiser la branche souhaitée au commit demandé.

```
$ git rm -rfv .*
$ git clone https://github.com/user/myrepo.git
```

----- ou -----

```
$ git reset --hard origin master
```



**BNP PARIBAS**

La banque d'un monde qui change

| 06/03/2020 | 49

BNPP Classification : Internal



# QUESTIONS



**BNP PARIBAS**

La banque d'un monde qui change

# Documentation

---

- **<https://git-scm.com/doc>**

C'est *la* documentation officielle de git. Vous y trouverez une liste exhaustive de toutes les commandes, de leurs options ainsi qu'un descriptive de leur fonction.

- **<http://ndpsoftware.com/git-cheatsheet.html#loc=;>**

Représentation visuelle des différentes commandes de Git.

- **<https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud>**

Documentation claire fournie, et illustrée des différentes commandes.

- **<https://app.pluralsight.com/library/courses/how-git-works/>**

Tutoriel en ligne synthétique (anglais, sous-titre français disponibles)

- **<https://gitignore.io>**

Pour vous aider à construire vos fichiers .gitignore



# Annexe : Aide Mémoire (1/2)

Actions	Commande
Comment récupérer un dépôt Git?	<code>git clone &lt;repository-url&gt;</code>
Comment ajouter un fichier au périmètre du prochain commit?	<code>git add &lt;file-name&gt;</code>
Comment faire un commit?	<code>git commit -m "&lt;message&gt;"</code>
Comment faire un commit en ajoutant tous les fichiers modifiés à son périmètre?	<code>git commit -am "&lt;message&gt;"</code>
Comment créer une branche?	<code>git checkout -b &lt;branch-name&gt;</code> <code>git branch &lt;branch-name&gt;</code>
Comment créer une branche à partir d'un tag?	<code>git checkout -b &lt;branch-name&gt; &lt;tag-name&gt;</code> <code>git branch &lt;branch-name&gt; &lt;tag-name&gt;</code>
Comment lister les branches?	<code>git branch -a</code>
Comment changer de branche?	<code>git checkout &lt;branch-name&gt;</code>
Comment fusionner une branche dans la branche sur laquelle on se trouve?	<code>git merge &lt;branch-name&gt;</code>
Comment supprimer une branche?	<code>git branch -d &lt;branch-name&gt;</code>
Comment mettre à jour ses références locales avec le dépôt central?	<code>Git fetch</code>
Comment mettre à jour ses références locales et intégrer les modifications du dépôt central sur la branche courante?	<code>Git pull --rebase</code>
Comment publier mes commits d'une branche sur le dépôt central?	<code>git push origin &lt;branch-name&gt;</code>



# Annexe : Aide Mémoire (2/2)

Actions	Commande
 Comment consulter l'historique?	<code>git log --oneline --decorate</code>
 Comment afficher l'état des fichiers du dépôt?	<code>git status</code>
 Comment afficher la différence entre 2 commits?	<code>git diff &lt;commit-id&gt; &lt;commit-id&gt;</code>
 Comment ajouter un tag?	<code>git tag &lt;tag-name&gt;</code>
 Comment publier un tag sur le serveur central?	<code>git push origin &lt;tag-name&gt;</code>
 Comment lister les tags?	<code>git tag -l -n3</code>
 Comment supprimer un tag?	<code>git tag -d &lt;tag-name&gt;</code>
 Comment annuler toutes mes modifications depuis mon dernier commit ?	<code>git reset --hard HEAD</code>
 Comment réinitialiser un fichier à l'état dans lequel il est dans le périmètre du prochain commit?	<code>git checkout -- &lt;file-name&gt;</code>
 Comment retirer un fichier du périmètre de votre prochain commit en conservant les modifications?	<code>git reset HEAD &lt;file-name&gt;</code>
 Comment annuler un commit déjà publié?	<code>git revert &lt;commit-id&gt;</code>
 Comment remplacer le dernier commit, non publié, par un nouveau commit?	<code>git commit --amend</code>
 Comment supprimer des commits non publiés en conservant les modifications sur son espace de travail?	<code>git reset --mixed &lt;commit-id&gt;</code>



# MERCI !



**BNP PARIBAS**

La banque d'un monde qui change