



Formation Git

Gilbert NZEKA



Qui suis-je?

◆ Entrepreneur / CTO

◆ Formateur

◆ Auteur







Quelles sont vos attentes?





Programme du cours

◇ Histoire de GIT

◇ Différences avec les autres solutions de gestion de code

◇ Les bases de GIT

◇ Branching & Merging

◇ Travailler à distance (pull/push)

◇ Exercices...





Histoire de Git

D'où vient Git ?



D'où vient Git?

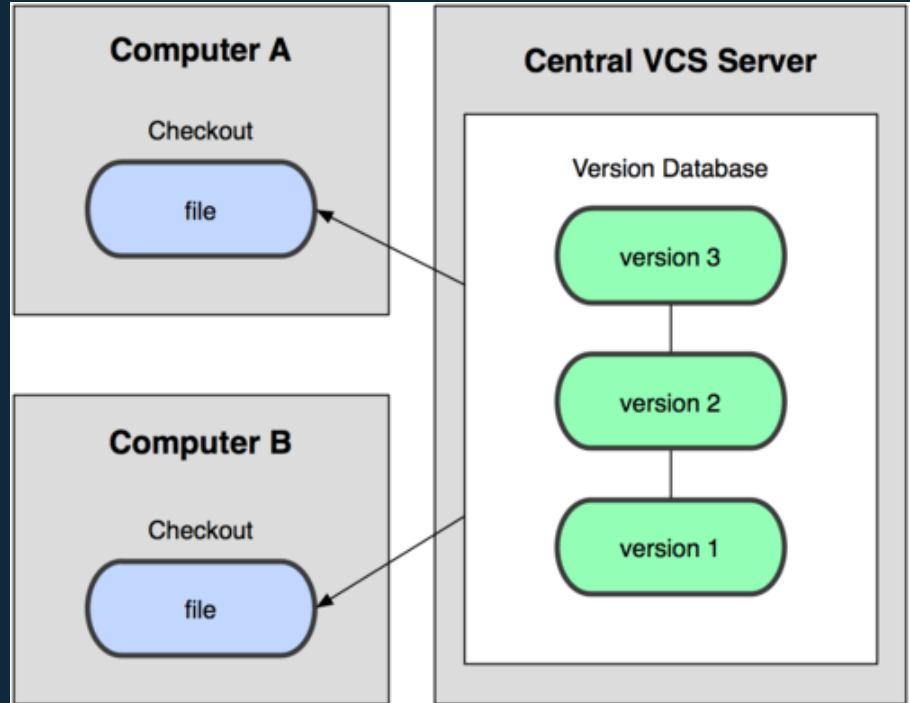
- ◇ Git est une solution de **gestion de version**
- ◇ C'est un système qui enregistre l'évolution d'un fichier ou d'un ensemble de fichiers au cours du temps de manière à ce qu'on puisse rappeler une version antérieure d'un fichier à tout moment
- ◇ VCS en anglais pour *Version Control System*
- ◇ Il existe 2 types : les VCS centralisés et les VCS distribués.





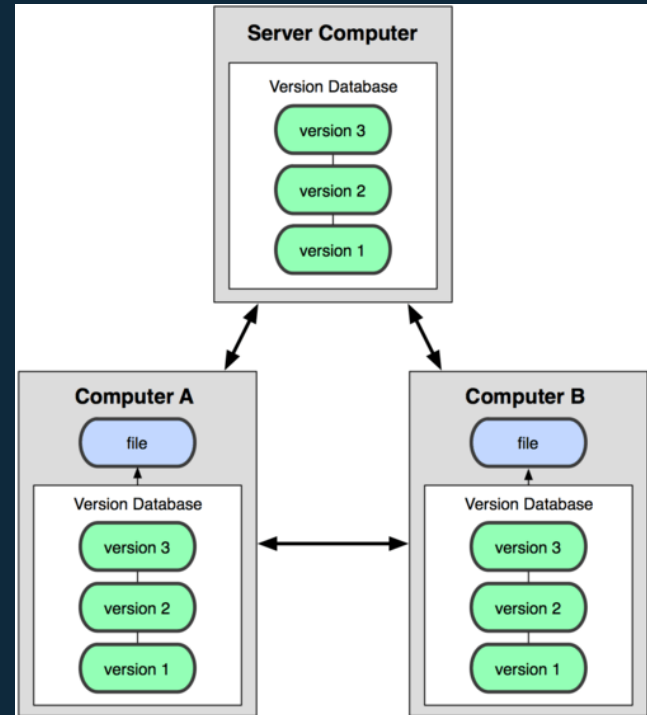
D'où vient Git?

Ces systèmes centralisés tels que CVS, Subversion, et Perforce, mettent en place un serveur central qui contient tous les fichiers sous gestion de version, et des clients qui peuvent extraire les fichiers de ce dépôt central.



D'où vient Git?

Dans le modèle décentralisé (tel que Git ou Mercurial), les clients n'extraient plus seulement la dernière version d'un fichier, mais ils dupliquent complètement le dépôt. Ainsi, si le serveur disparaît et si les systèmes collaboraient via ce serveur, n'importe quel dépôt d'un des clients peut être copié sur le serveur pour le restaurer.





D'où vient Git?

- ◆ Le développement de Git est lié au noyau LINUX. Le noyau Linux est un projet libre de grande envergure. En 2002, le projet du noyau Linux commença à utiliser un DVCS propriétaire appelé BitKeeper.
- ◆ Mais en 2005, Linus Torvalds, le créateur de Linux, a décidé de développer son propre outil.





D'où vient Git?

- ◇ Depuis sa naissance en 2005, Git a évolué et mûri pour être facile à utiliser tout en conservant ses qualités initiales. Il est incroyablement rapide, il est très efficace pour de grands projets et il a un incroyable système de branches pour des développements non linéaires







Codons à plusieurs !

Versionner son projet avec Git



Pourquoi utiliser Git?

Objectifs :

- ◆ **travailler à plusieurs sans se marcher dessus** : indispensable pour les projets en équipe
- ◆ **garder un historique propre de toutes les modifications** : on organise son travail sous forme de "commits" documentés





Pourquoi utiliser Git?

Objectifs :

- ◇ Vitesse
- ◇ Rapidité
- ◇ Complètement distribué
- ◇ Support pour les développements non linéaires (milliers de branches parallèles)
- ◇ Capacité à gérer efficacement des projets d'envergure tels que le noyau Linux (vitesse et compacité des données)





La théorie de git

3 zones, 3 ambiances

Les modifications sont sauvegardés 3 fois

Working directory

C'est la zone de travail : les fichiers tout juste modifiés sont ici

Index

Zone qui permet de stocker les modifications sélectionnées en vue d'être commitées

Local repository

Code commité, prêt à être envoyé sur un serveur distant



Les commits

Commit : ensemble de modifications cohérentes du code

Un bon commit est un commit :

- ◇ qui ne concerne qu'une seule fonctionnalité du programme
- ◇ le plus petit possible tout en restant cohérent
- ◇ Idéalement qu'il compile seul

C'est quoi concrètement un commit ?

- ◇ une différence (ajout / suppression de lignes)
- ◇ des méta-données (titre, hash, auteur)

Les commits

Arbre de commits dans le git repository

4a7f5a6c478..


7a4d5f97d8bb..

0a4d9f8d4dda..

Formulaire ajout
commentaire

Ajout
commentaire
dans base de
données

Affichage des
commentaires
des publications



Aller jusqu'au commit

3 zones, 3 ambiances

Les modifications sont sauvegardés 3 fois

Working directory

C'est la zone de travail : les fichiers tout juste modifiés sont ici

Index


Zone qui permet de stocker les modifications sélectionnées en vue d'être commitées

Local repository

Code commité, prêt à être envoyé sur un serveur distant

git add

git
commit



Aller jusqu'au commit

Où j'en suis dans mes 3 zones ?

```
git status
```

```
On branch master
```


```
Changes not staged for commit:
```

```
(use "git add <file>..." to update what will be committed)
```

```
(use "git checkout -- <file>..." to discard changes in working directory)
```

```
    modified:   views/add_commentaire.php
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```




Aller jusqu'au commit

Visualiser les différences entre le working directory et l'index

```
git diff
```

```
diff --git a/views/add_commentaire.php b/views/add_commentaire.php
index e69de29..8cff573 100644
--- a/views/add_commentaire.php
+++ b/views/add_commentaire.php
@@ -0,0 +1,6 @@
<?php include('header.php'); ?>

+<form action="/commentaire/ajouter" method="post">
+    <p>Pseudo : <input type="text" name="pseudo"/></p>
+    <p>Commentaire : <textarea name="commentaire"></textarea></p>
+</form>
```



Aller jusqu'au commit

Ajouter mes modifications à la zone de staging
(index)


```
git add views/add_commentaire.php  
git status
```

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

```
modified:   views/add_commentaire.php
```



Aller jusqu'au commit

Récapitulatif des commandes

affichage
différences

`git diff`

`git diff --staged`

working directory

index

git repository

ajouter des
modifications

```
git add mon_fichier  
git add -p (interactif)  
git add -A (tout ajouter)
```

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



Revenir au dernier commit

```
git status
```

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: model/commentaires_model.php

no changes added to commit (use "git add" and/or "git commit -a")



Revenir au dernier commit

Enlever des modifications dans le working directory

```
git checkout -- monfichier  
git status
```

On branch master
nothing to commit, working directory clean



Désindexer des fichiers

```
git status
```

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

```
modified:   model/commentaires_model.php
```



Désindexer des fichiers

```
git reset HEAD monfichier  
git status
```

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: model/commentaires_model.php

no changes added to commit (use "git add" and/or "git commit -a")



Aller jusqu'au commit

Récapitulatif des commandes

working directory

index

git repository

`git diff`

`git diff --staged`

`git add monfichier`

`git commit -m "titre"`

`git commit -am "titre" (si le fichier a déjà été indexé)`

`git checkout -- monfichier`

`git reset HEAD monfichier`

`git reset --hard`



Mise en pratique

Récapitulatif des commandes

EXERCICE

Git gère vos informations dans un dépôt (repository), qui correspond en réalité à un répertoire caché de votre répertoire de travail (celui qui contient les fichiers dont vous voulez conserver différentes versions).

```
git init;
```

Lorsque git enregistre les informations que vous lui soumettez, git ajoute le nom de la personne qui demande l'enregistrement (la sauvegarde). Cela permet, lorsqu'on travaille dans un environnement multi-collaborateurs, de retrouver qui a enregistré quoi.

```
git config --global user.name VotreNom;
```

```
git config --global user.email VotreAdresseMail;
```



Mise en pratique

Récapitulatif des commandes

EXERCICE

Vous devez indiquer à git quel est l'ensemble des changements dans vos fichiers que vous voulez qu'il conserve pour la prochaine fois que vous demanderez la sauvegarde. Pour cela, vous indiquez à git de placer vos changements dans une zone temporaire (qu'on appelle l'index, et dont le contenu est conservé jusqu'à la prochaine sauvegarde)

```
touch NomDuFichier;  
git add NomDuFichier;
```

La commande git status affiche la situation de chacun des fichiers de votre répertoire de travail (et fichiers des sous-répertoires éventuellement), et vous donne aussi des indices sur ce que vous pourriez faire pour faire gérer vos données.

```
git status;
```



Mise en pratique

Récapitulatif des commandes

EXERCICE

Enregistrer vos informations dans le dépôt. Lorsque vous enregistrez un ensemble de changements (contenus dans vos fichiers), git enregistre ce qui est dans l'index, et vous demande de décrire en quoi concernent les changements.

```
git commit -m "Message qui décrit les changements";
```

Connaitre l'historique des commits de votre projet. La commande git log affiche, en ordre chronologique inversé, chaque commit réalisé sur le dépôt, l'heure à laquelle il a été créé, le nom de la personne qui l'a effectué et la première ligne du message de description de ce commit.

```
git log;
```

A vous de tester les autres commandes vues en cours !!!





Les branches !

Versionner son projet avec Git



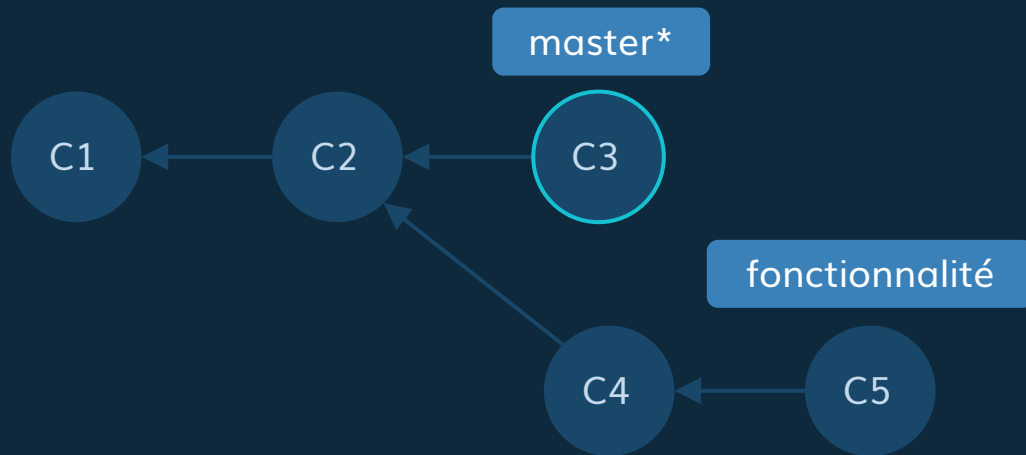
Les branches

- ◇ branche : *pointeur* vers un commit
- ◇ une branche principale : **master**
- ◇ Branche courante : **HEAD**
- ◇ en général, une branche par fonctionnalité en cours de développement



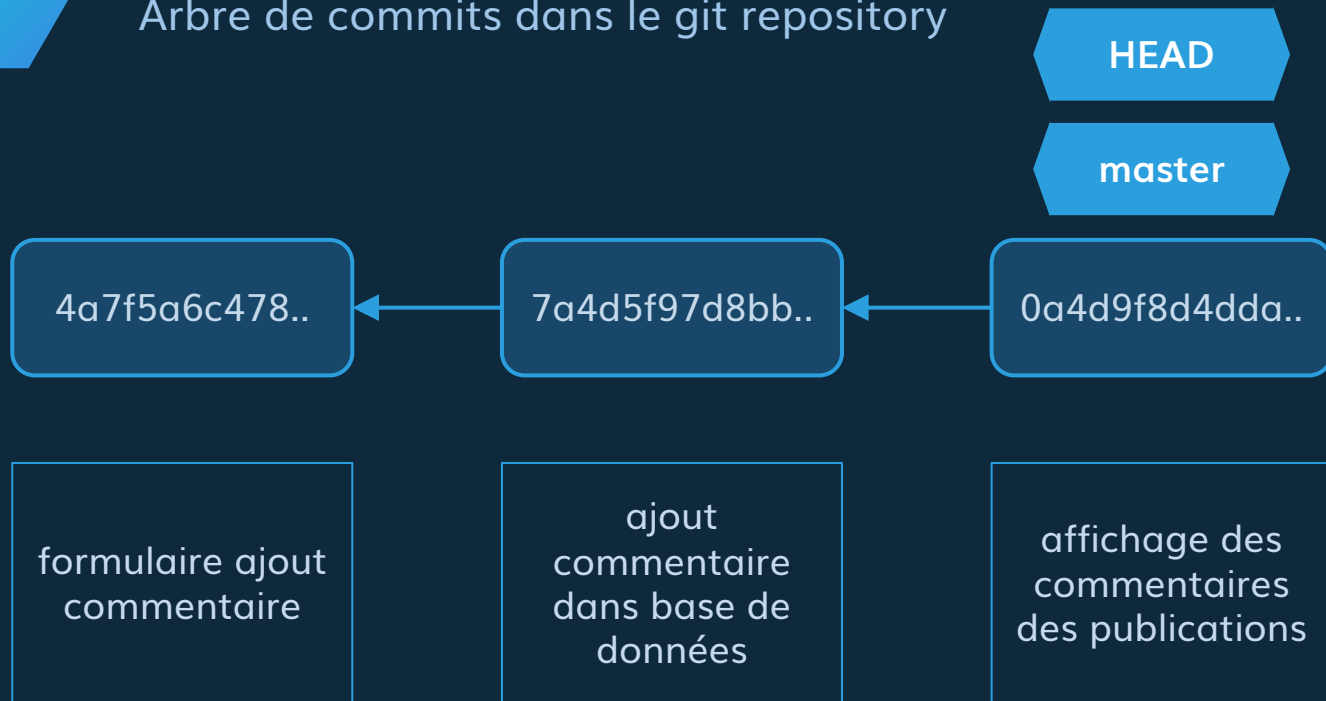


Les branches



Les branches et commits

Arbre de commits dans le git repository





Gestion des branches

Création et modification de branches

```
git branch : affichage des branches
```

```
git branch ma_branche : créer la branche ma_branche
```

```
git checkout ma_branche : déplace HEAD vers ma_branche
```

ou pour simplifier...

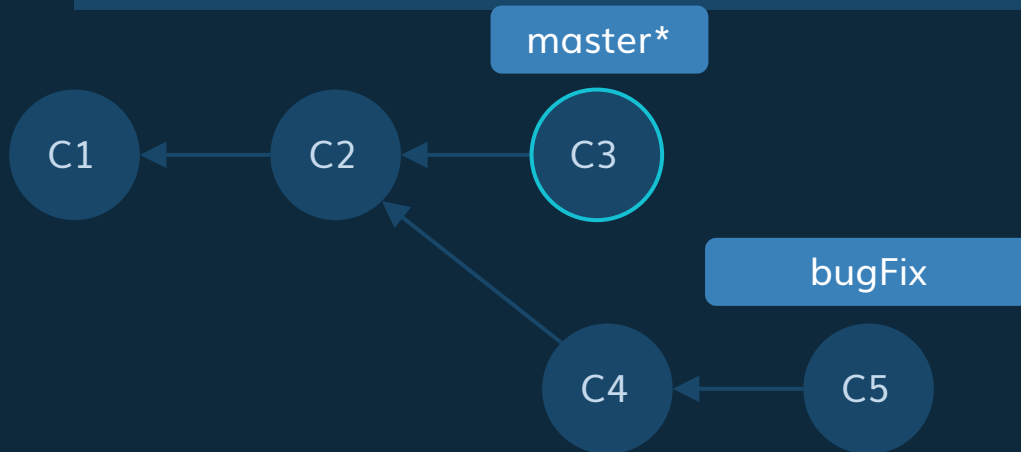
```
git checkout -b ma_branche : créer la branche  
ma_branche et déplace HEAD dessus
```



Gestion des branches

Merge : intégration des modifications d'une branche dans la branche courante

```
git merge ma_branche: merge ma branche dans la branche courante
```

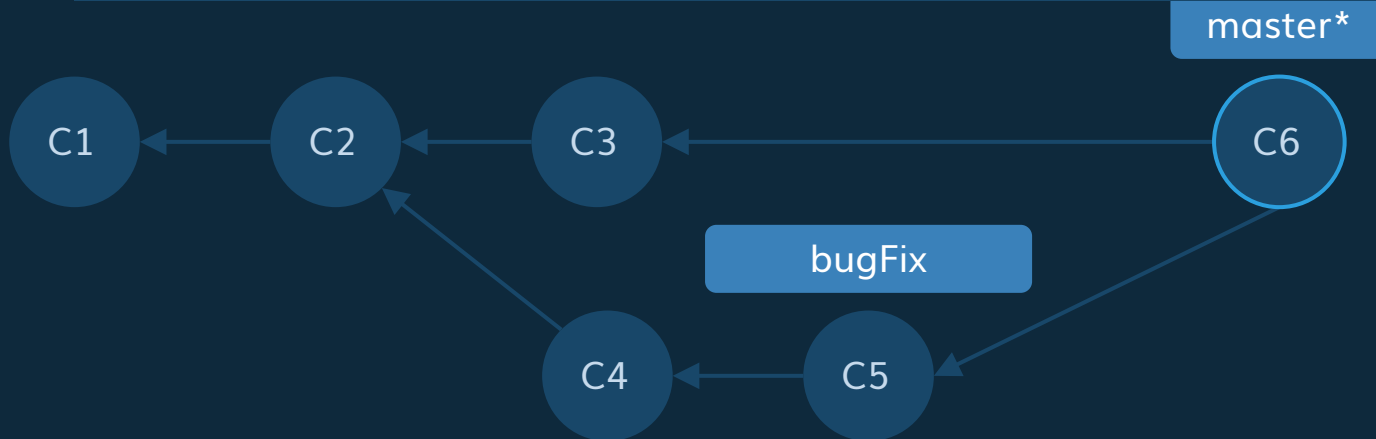




Gestion des branches

Merge : intégration des modifications d'une branche dans la branche courante

```
git merge ma_branche: merge ma branche dans la branche courante
```






Gestion des conflits

Quelques fois, le processus ci-dessus ne se passe pas sans accroc. Si vous avez modifié différemment la même partie du même fichier dans les deux branches que vous souhaitez fusionner, Git ne sera pas capable de réaliser proprement la fusion.

```
$ git merge prob53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```





Gestion des conflits

Dans le ou les fichiers posant problèmes, vous aurez des annotations du genre :

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>> prob53:index.html
```





Gestion des conflits

A vous d'ouvrir le ou les fichiers et corriger le conflit manuellement puis refaire un git add.

```
<div id="footer">
please contact us at email.support@github.com
</div>
```







Les dépôts distants

Centraliser les données sur un dépôt git !



Les dépôts distants

- ◇ Git directory sur un serveur distant pour le travail collaboratif
- ◇ Dépôts distants :
 - Github





Voir et ajouter des dépôts distants

Cloner un dépôt distant : crée un dossier et récupère les fichiers

```
git clone <url>
```



Envoyer sur le dépôt distant

```
git push
```

Envoie notre local repository sur le dépôt distant

On ne touche plus aux commits pushés !



Recevoir depuis le dépôt distant

```
git pull
```

Récupère les commits sur le dépôt distant et met à jour le working directory



Le schéma de base de git

Récapitulatif des commandes

working directory

index

git repository

remote git
repository

`git diff`

`git diff --staged`

`git add`

`git commit`

`git checkout --`

`git reset`

`git reset --hard`

`git push`

`git pull`

`git fetch`






Débuter avec Github



Débuter avec Github

GitHub est une plate-forme web de contrôle de version et de collaboration pour les développeurs de logiciels.

GitHub facilite le travail en équipe en fournissant une interface web afin d'accéder au dépôt de code Git et des outils de gestion pour la collaboration. Les membres peuvent se suivre les uns les autres, évaluer le travail des autres, recevoir des mises à jour pour des projets spécifiques et communiquer publiquement ou en privé.





Débuter avec Github

GitHub permet aux développeurs de changer, d'adapter et d'améliorer les logiciels de ses dépôts publics gratuitement, mais il est nécessaire de payer pour les dépôts privés. Chaque dépôt public ou privé contient tous les fichiers d'un projet, ainsi que l'historique des révisions de chaque fichier.



Débuter avec Github



GitHub




Mise en pratique

Récapitulatif des commandes

EXERCICE

```
git clone https://github.com/nachosagency/FormationGit.git  
git remote add origin https://github.com/nachosagency/FormationGit.git  
git push origin master  
git pull origin master
```



A decorative graphic on the left side of the slide. It features a large, solid blue hexagon in the center. Surrounding it are several smaller hexagons of varying shades of blue and cyan. Some of these hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, a gear, and a speech bubble. There is also a small network diagram icon with a central node and five connecting lines.

Différences entre git rebase & git merge



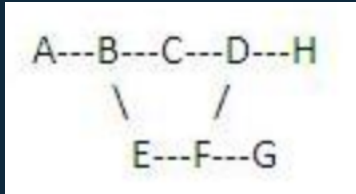
Différences entre git rebase & git merge

La première chose à savoir sur la commande *git rebase* est qu'elle poursuit le même objectif que *git merge*. Ces deux commandes permettent de transférer des changements d'une branche à une autre. Seule la manière de procéder diffère.



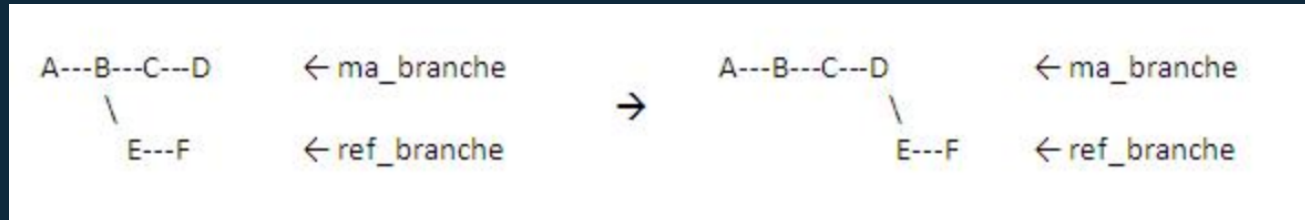
Différences entre git rebase & git merge

Le Merge permet d'avancer la branche courante en incorporant le travail d'une autre branche. Cette opération joint et fusionne les deux points de départ de ces deux branches dans un commit de fusion qui est visible dans le graphe de l'historique



Différences entre git rebase & git merge

Le Rebase rejoue sur une branche (que l'on appelle `ma_branche`) les commits d'une autre branche (que l'on appelle `ref_branche`) dans l'ordre d'apparition à partir du point de séparation. C'est comme si tous les travaux de `ref_branche` avaient été réalisés sur `ma_branche`. Le Rebase est transparent dans l'historique et permet de conserver un graphe linéaire.





Différences entre git rebase & git merge

Dans quel cas utiliser un rebase?

Quand un travail local est considéré comme partant d'une base obsolète. On a travaillé en local sur une base sans faire un pull avant. Le push nous enverrait balader et faire un merge ajouterait trop de bruit. Il faut donc faire un git pull --rebase.





Différences entre git rebase & git merge

Dans quel cas utiliser un merge?

La branche xxxxxx représente un sprint ou une story en méthodologie agile, ou encore un ticket d'incident (issue ou bug) précis, identifié dans la gestion de tâches. Il est alors préférable d'utiliser la méthode merge. Cela conserve l'historique du bug.





Différences entre git rebase & git merge

Dans quel cas utiliser un merge?

La branche xxxxxxxx est dédiée pour la correction des anomalies sur la production. Il faut alors utiliser la méthode merge car il est difficile de faire un rebase si plusieurs corrections sont réalisées sur un même périmètre.





Mise en pratique

Récapitulatif des commandes

EXERCICE

```
git checkout master  
git merge experiment  
  
git checkout experiment  
git rebase master
```







Annexe : git show



Mise en pratique

Récapitulatif des commandes

EXERCICE



```
# Montre les modifications de métadonnées et de contenu incluses dans le commit  
spécifié  
git show [commit];
```





Annexe : git blame



Annexe : git blame

La commande 'git blame' est vraiment utile pour savoir qui a modifié quelle partie du fichier. Si vous lancez simplement 'git blame [nom_du_fichier]' vous obtiendrez une liste du dernier SHA du commit, de la date et de l'auteur de chaque ligne du fichier.





Mise en pratique

Récapitulatif des commandes

EXERCICE

`git blame` #is used to know who/which commit is responsible for the latest changes made to a file, author/commit of each line can also be seen.

`git blame filename` #(commits responsible for changes for all lines in code)

`git blame filename -L 0,10` #(commits responsible for changes from line "0" to line "10")

#There are many other options for blame, generally these could help





Annexe : git log & git reflog



Mise en pratique

Récapitulatif des commandes

EXERCICE

Montre l'historique des versions pour la branche courante
`git log ;`

Montre l'historique des versions, y compris les actions de renommage, pour le
fichier spécifié
`git log --follow [fichier]`



Mise en pratique

Récapitulatif des commandes

EXERCICE

Pendant que vous travaillez, Git enregistre l'emplacement de votre HEAD chaque fois que vous le changez. À chaque commit ou commutation de branche, le journal des références (reflog) est mis à jour. Pour accéder au SHA pour plus tard recréer une branche perdue (par exemple)

```
git reflog ;
```



A decorative graphic on the left side of the slide consists of a large central cyan hexagon. Surrounding it are several smaller hexagons of varying shades of blue and cyan. Some of these smaller hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, and a gear. There is also a network-like icon with a central node and radiating lines, and a speech bubble icon.

Annexe : Supprimer un fichier de l'historique



Mise en pratique

Récapitulatif des commandes

EXERCICE

Vous avez envoyé un mot de passe en clair dans Git? Vous êtes une startup et vous souhaitez supprimer de "mauvais" agissements?

```
git filter-branch --force --index-filter 'git rm --cached --ignore-unmatch  
chemin/vers/le/fichier.txt' --prune-empty --tag-name-filter cat -- --all
```





Projet final !

A decorative graphic on the left side of the slide. It features a large, central cyan hexagon. Surrounding it are several smaller hexagons in various shades of blue and cyan. Some of these hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, and a gear. There is also a network-like icon with a central node and connecting lines, and a speech bubble icon.

Installons le Git (en prod)
pour votre projet !

