FORMATION GIT

SYSTÈME DE GESTION DE VERSION DÉCENTRALISÉ

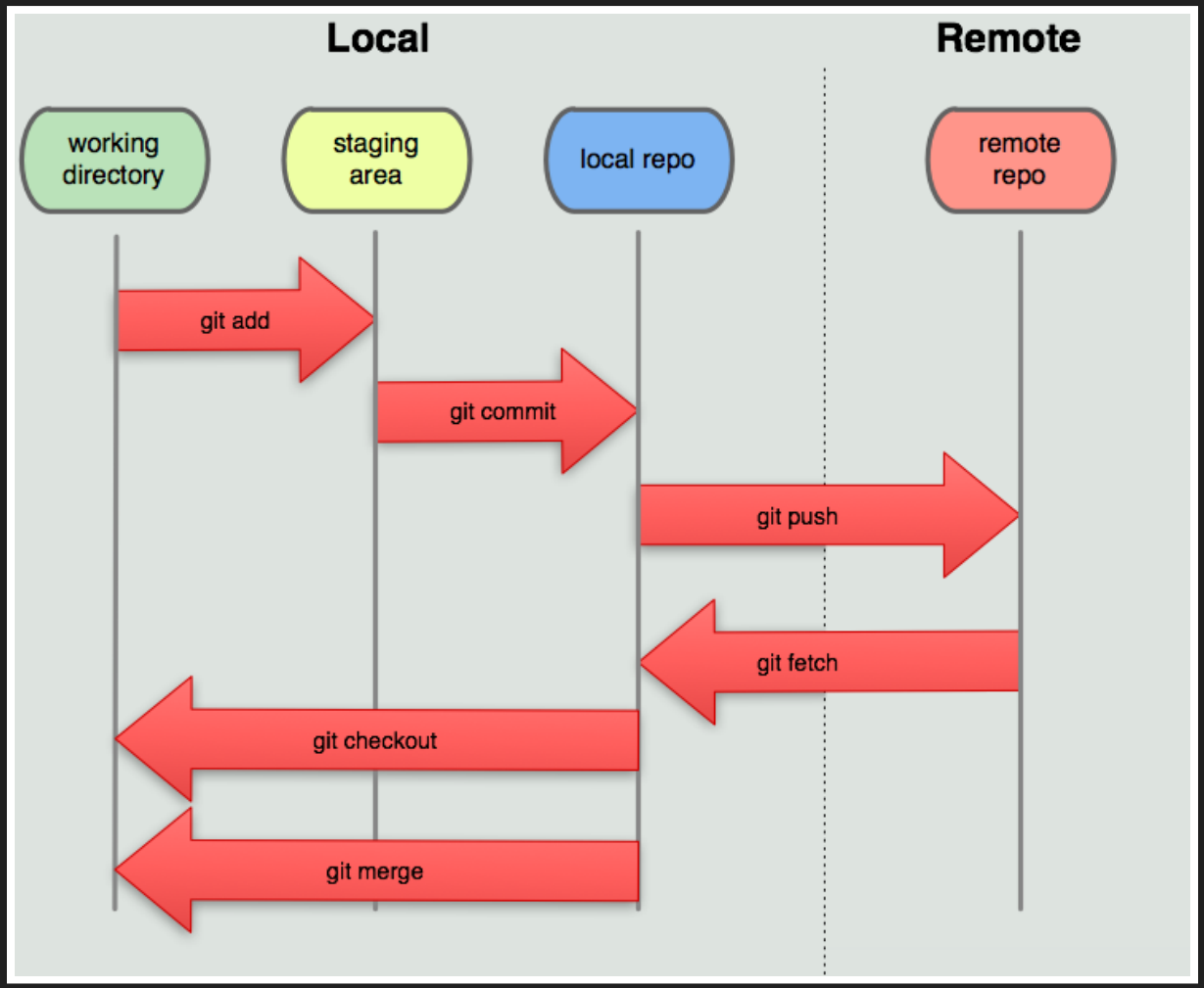
Partie 2 - Travail collaboratif

Présenté par Jonathan Martel

PLAN DE CONTENU

- Retour sur la base
- Collaboration avec Git
- Méthode de travail en équipe(*workflow*)

RÉVISION DE LA BASE



```
# Cloner un repository (juste si le dossier n'est pas déjà en local)
> git clone URL_DU_DEPOT_DISTANT

# Vérifier l'état du dépot (repository)
> git status

# Ajout du fichier à l'index (staging)
> git add fichier.ext
> git add .              #Pour tous les fichiers modifiés

# Faire un commit
> git commit -m "Message clair"

# Permet d'ajouter les fichiers modifiés et effacés automatiquement
# au staging (sauf nouveaux fichiers)
> git commit -a -m "Message clair"
```

```
# Vérifier que le dépôt distant et local sont à jour
> git remote update      # Met à jour les informations du dépot
> git status    # Affiche les différences entre le local et le distar
# Seulement si la différence entre local et distant n'apparait pas
> git push -u origin master

# Récupérer les modifications
> git fetch origin master   #Recupère les données distantes
> git branch -r             #Liste des branches distantes
> git merge origin/master   #Fusionne les branches
# Alternative
> git pull origin master    # fetch et merge ensemble

# Envoyer les commits sur le distant
> git push origin master    # Envoie les modifications vers le distar
```

EN BREF :

Avant de travailler

```
> git remote update      # Met à jour les informations du dépot
> git status
# Si remote est en retard (moins nécessaire)
> git push origin master # Met le dépôt distant à jour

# Si local est en retard (très important)
> git pull origin master # Met le dépôt local à jour
```

travail, travail, travail...

```
> git add fichiers_modifies # ou "git add .", pour tous les fichiers
> git commit -m "Voici ce que j'ai fait"
```

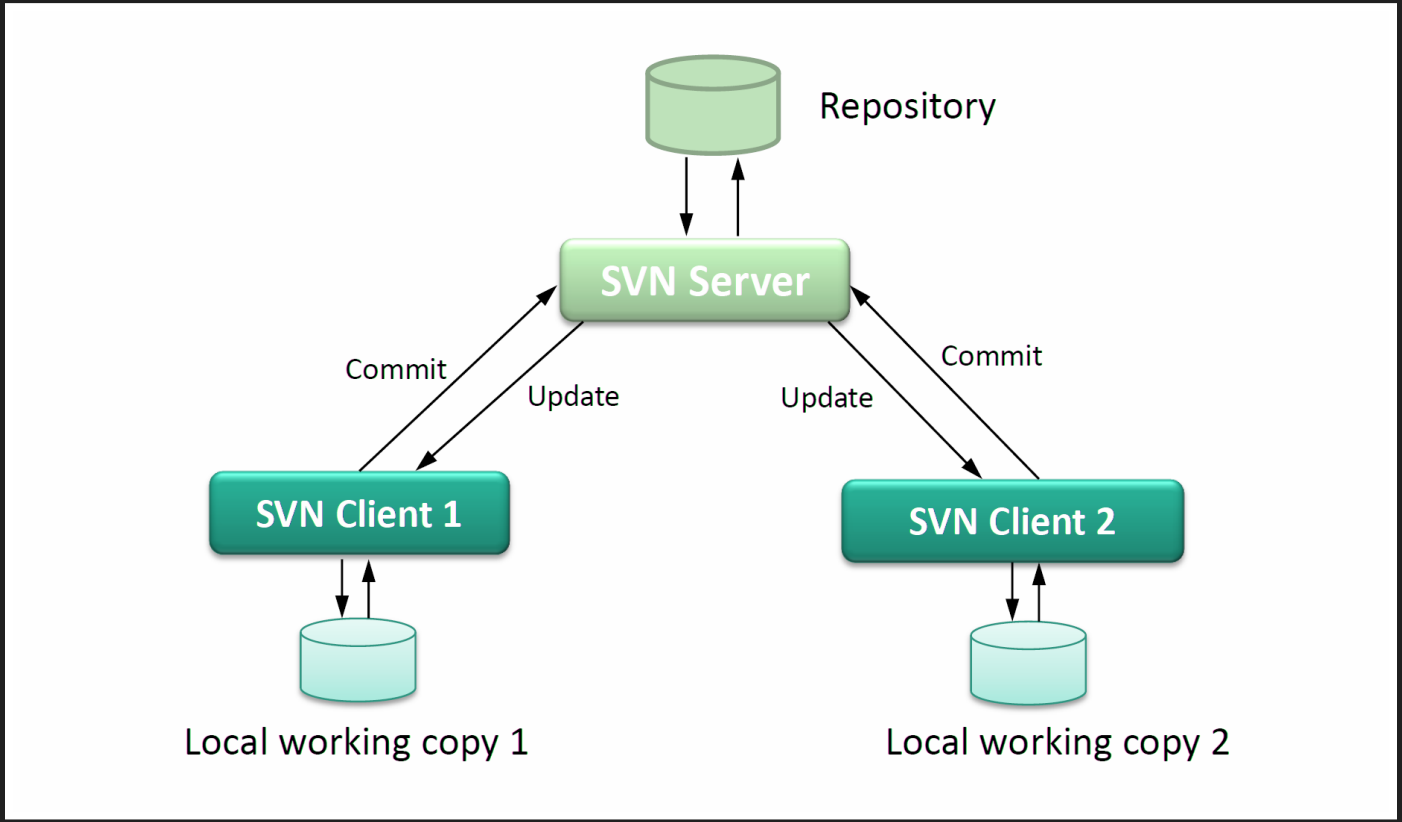
...jusqu'à la fin d'une période de travail

```
> git add .
> git commit -m "Voici ce que j'ai ajouté ou ce que j'ai corrigé"
> git push origin master # Met le dépôt distant à jour
```

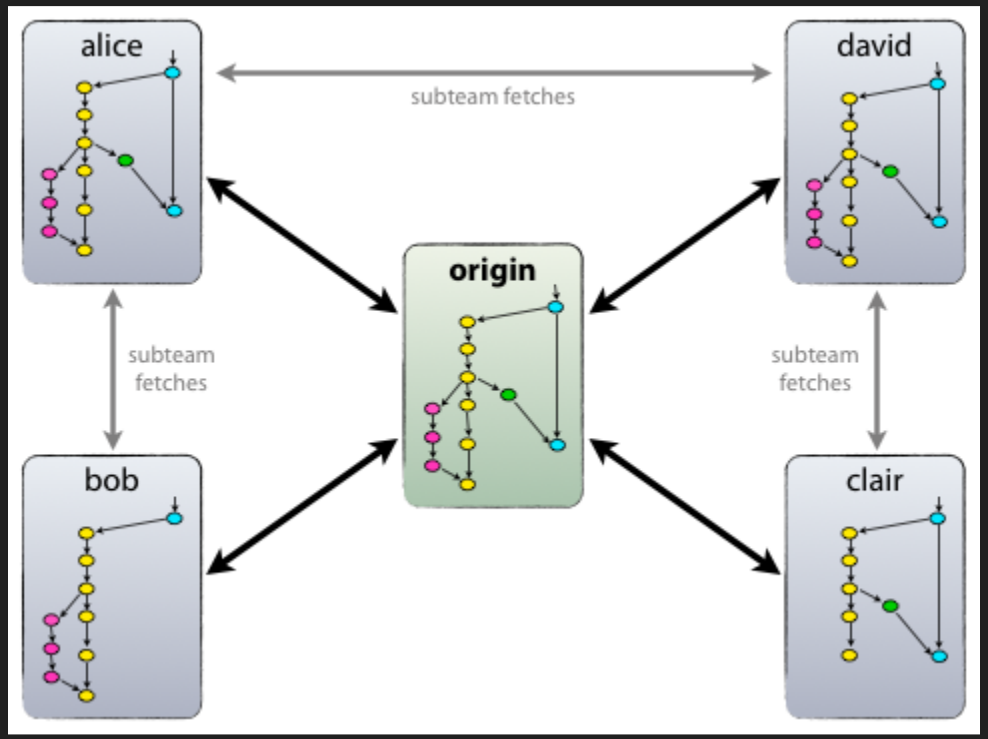
GIT EN ÉQUIPE

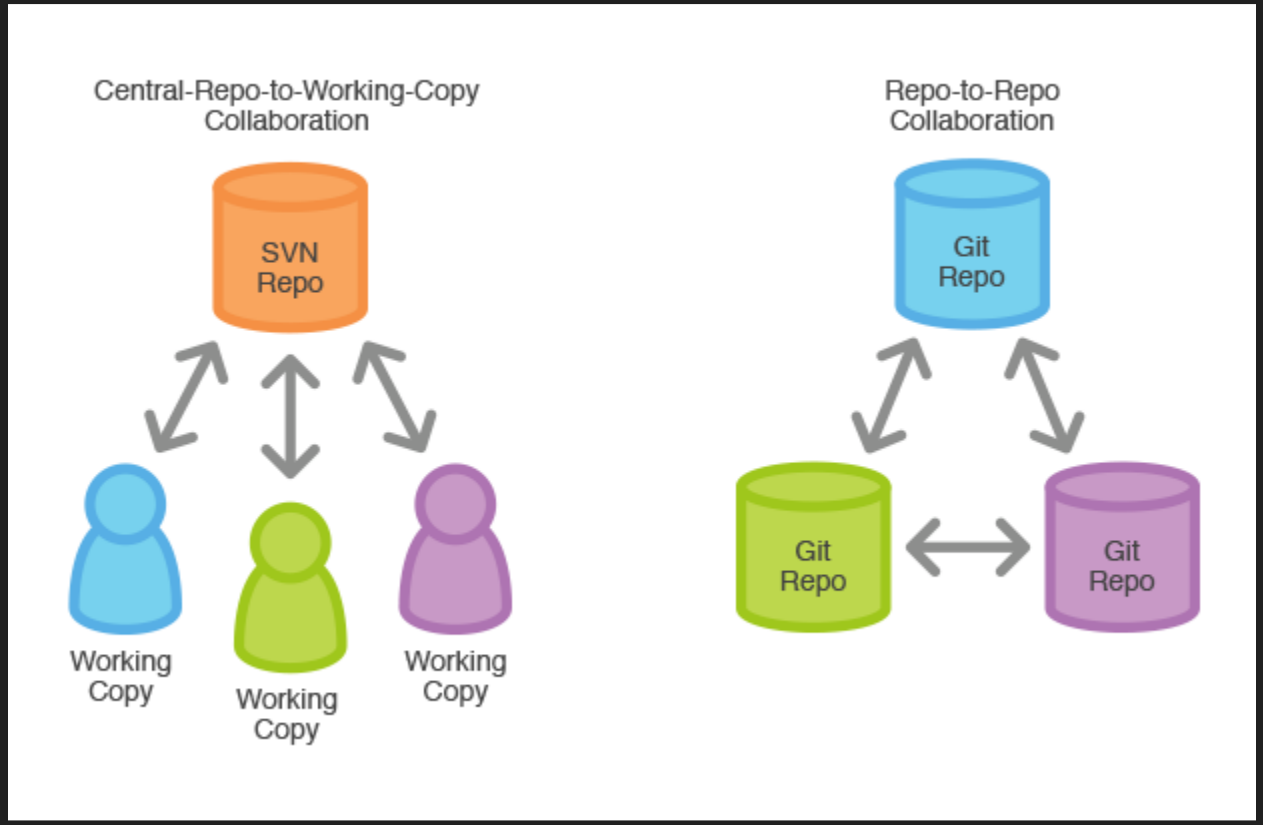
- Permet de favoriser le partage de ressources
- Outils de coordination et de suivi des développements
- Architecture décentralisée et sécuritaire

ARCHITECTURE D'UN PROJET UTILISANT UN DÉPÔT CENTRALISÉ



ARCHITECTURE D'UN PROJET GIT




Central-Repo-to-Working-Copy Collaboration
SVN Repo
Working Copy
Working Copy
Working Copy
Repo-to-Repo Collaboration
Git Repo
Git Repo
Git Repo

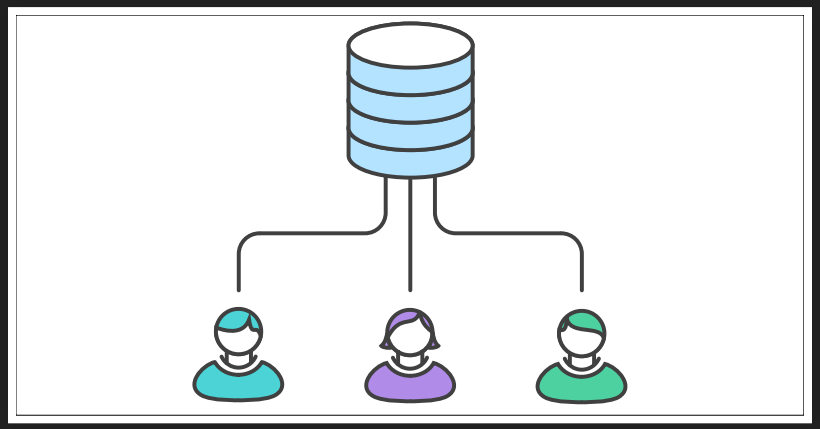
SVN VS GIT

- SVN est vulnérable à la corruption du code
- SVN demande une planification de backup plus avancées
- SVN fait plus de charge sur le réseau
- SVN exige plus d'espace disque
- SVN dépend de l'intégrité complète du projet
- SVN restreint le travail collaboratif "non centralisé"

DIFFÉRENTS *WORKFLOW* GIT

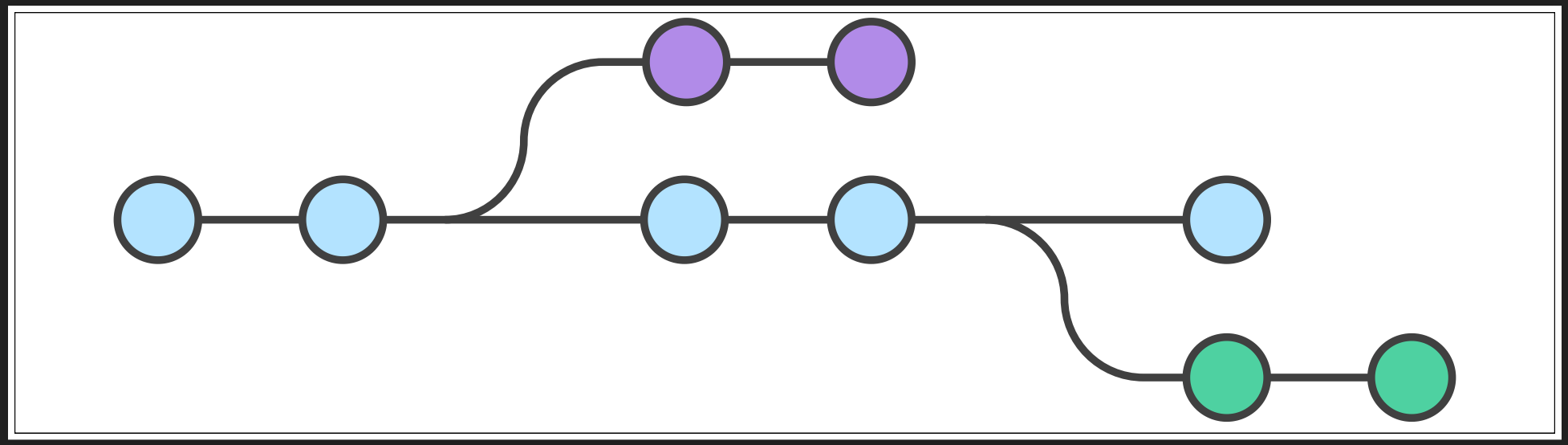
- Dépot centralisé
- Par branches de fonctionnalités/développeurs
- Forking workflow
- Gitflow - Vincent Driessen
- Gitlab flow

DÉPÔT CENTRALISÉ



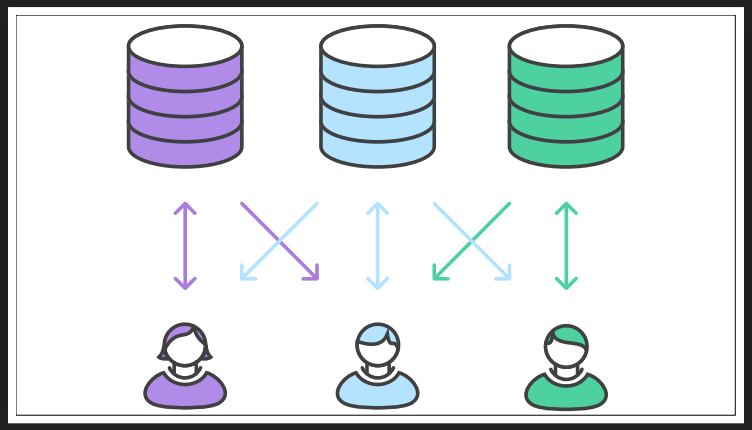
Le processus de travail est similaire à celui de SVN. Il consiste à travailler avec un seul dépôt central qui est distant (*remote*) et à mettre à jour à tour de rôle le dépôt. À chaque mise à jour, les autres récupèrent les mises à jour avant de soumettre les leurs.

PAR BRANCHES DE FONCTIONNALITÉS/DÉVELOPPEURS



À partir d'un dépôt centralisé, chaque fonctionnalité est développée sur une branche spécifique par les développeurs. Chacun travaille sur une branche unique à lui.

FORKING WORKFLOW

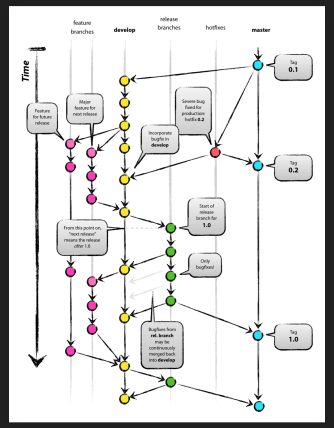


Chaque développeur *fork* le projet sur son propre dépôt distant. Ensuite il développe les fonctionnalités en utilisant un modèle de branche personnalisé (au besoin). Lorsqu'il a terminé, il fait une requête de fusion sur le dépôt initiale (*upstream*). Le chef du projet ou les développeur séniors font la fusion sur le dépôt initial.

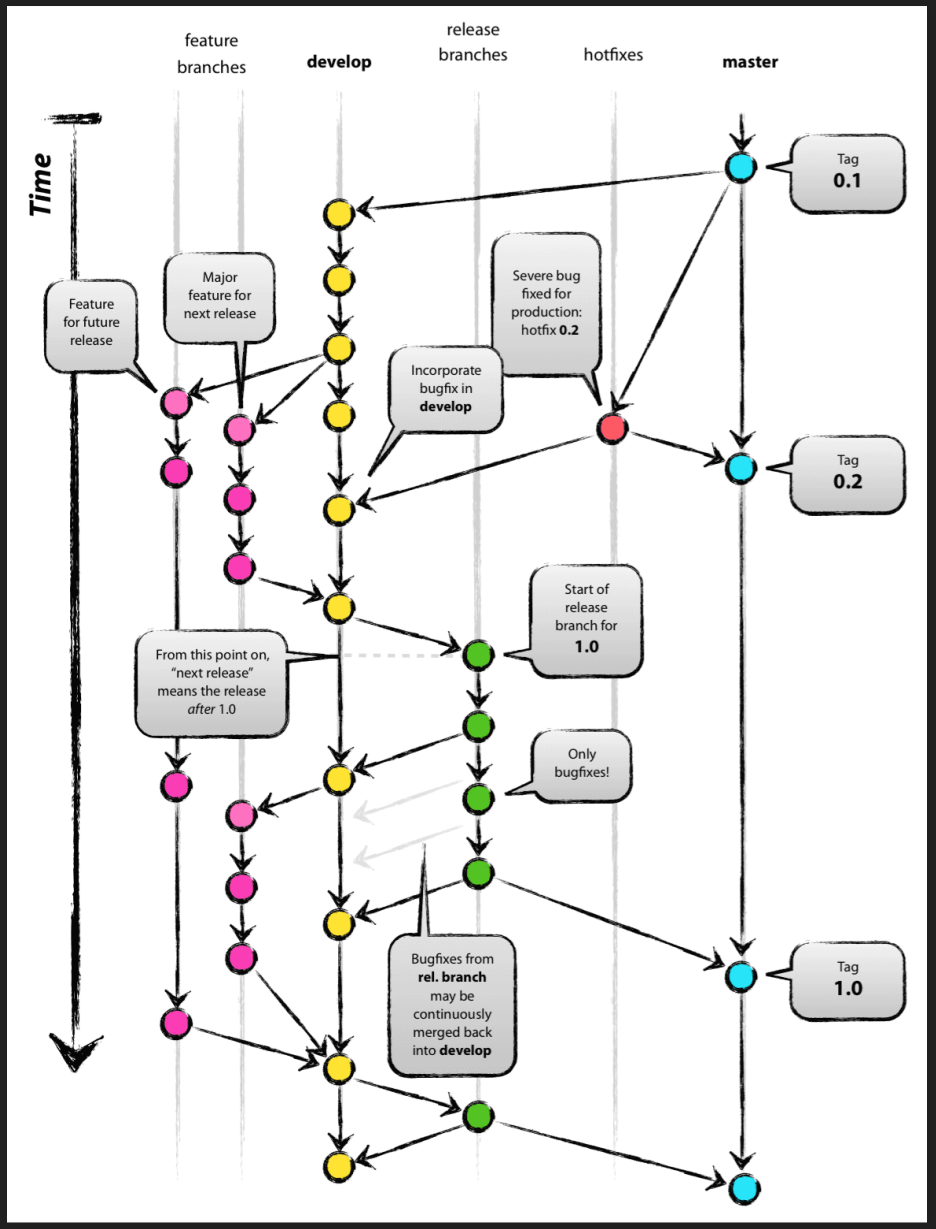
- Chaque développeur *fork* le dépôt à partir de l'interface du serveur, ils possèdent donc tous un dépôt distant individuel (privé)
- Ils acquièrent leur dépôt distant (*clone*)
- Les développeurs travaillent comme bon leur semble (*branch, add, commit, push, merge*, etc)
- Quand tout fonctionne, ils créent une requête de fusion sur le serveur (*pull request* sur github ou bitbucket)) avec le dépôt initiale (*upstream*) pour une branche spécifique

- Selon les droits du développeur, il peut lui-même gérer la requête et fusionner la branche ou bien un autre fera le travail de fusion (après correction et discussion avec le développeur)
- Une fois le dépôt initial mis à jour, les autres développeurs récupèrent la mise à jour (*fetch*) et fusionne leurs branches de développement locales (*merge*) et distantes (*push*)
- Personne ne travaille directement sur le dépôt central (*upstream*)

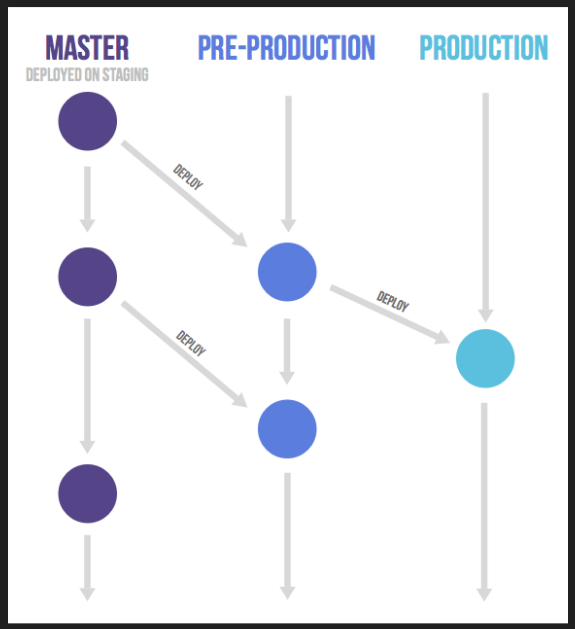
GITFLOW - VINCENT DRIESSEN



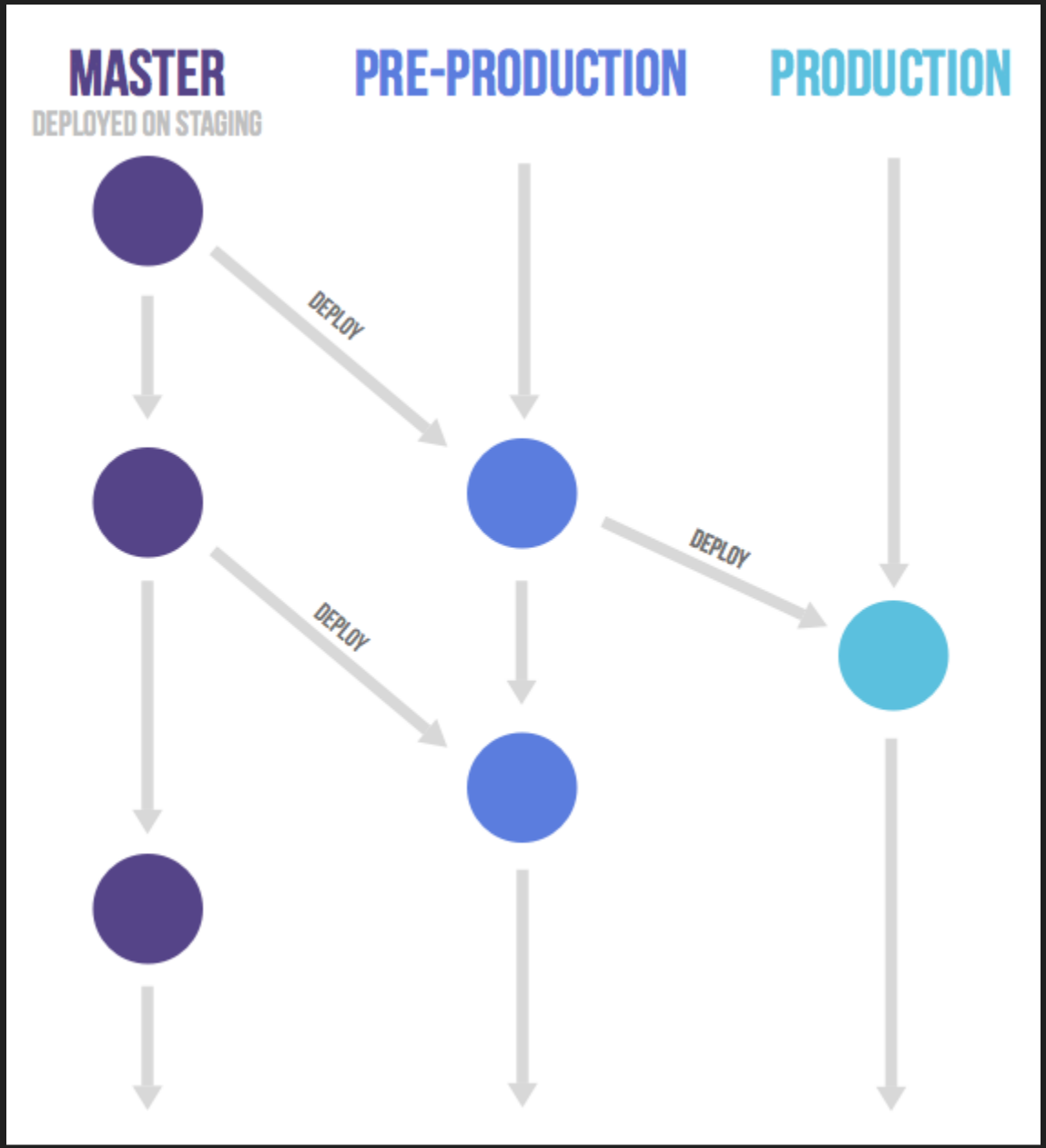
Gitflow est un modèle de travail qui peut s'appliquer aux autres structures (centralisées, branche de fonction et forking). Il consiste à travailler sur de multiples branches et réserver la branche master pour les versions finales (*release*).


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Major feature for next release
Feature for future release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

GITLAB FLOW



Modèle de développement proposé par Gitlab. Il propose trois branches et un modèle de travail linéaire (dev, pre-prod, prod) qui convient à de nombreux projets


MASTER
DEPLOYED ON STAGING
PRE-PRODUCTION
PRODUCTION
DEPLOY
DEPLOY
DEPLOY

CONFIGURATION NÉCESSAIRE POUR LE TRAVAIL COLLABORATIF

1. Créer ou avoir un compte github par membre
2. Créer une organisation (github)
3. Ajouter les membres à l'équipe (avec les droits)
4. Initialiser le dépôt de l'équipe (vide, à partir d'un fork ou d'un dépot local existant) (upstream)
5. Chaque membre "fork" le dépôt de l'équipe (upstream)
6. Chaque membre clone leur dépot distant sur le local
7. Configurer le upstream sur le dépôt local
8. Vous êtes maintenant prêt à travailler et partager votre code (code, add, commit, push, pull, etc)!

RÉCUPÉRER LA DERNIÈRE MISE À JOUR DU PROJET

1. S'assurer que tous les changements sont commités
2. S'assurer que le local et le remote sont synchronisés (pull/push avec origin)
3. Faire un pull de l'upstream
4. Corriger les conflits et finaliser la fusion (add, commit, merge)
5. Tester le code
6. Mettre à jour son remote

```
> git status    # Pour s'assurer que le dépot local est propre

# Récupérer les modifications de upstream
> git pull upstream master   #Recupère les données distantes de upstr

# Si le merge ne se fait pas automatiquement,
# réglez les conflits dans le code
# Testez et dépannez le code
# Quand tout est fonctionnel, terminé la fusion
> git add .
> git commit

# mettre à jour son remote
> git push origin master    # Envoie les modifications vers le distar
```

AJOUTER SON CODE AU PROJET DE L'ÉQUIPE

1. S'assurer que le local et le remote sont synchronisés (pull/push avec origin)
2. Récupérer la dernière mise à jour de upstream (pull upstream)
3. Corriger les conflits, le cas échéant (add, commit, merge, push)
4. Tester le fonctionnement du site (important!)
5. Mettre à jour son remote (git push)
6. Sur Github : créer une demande de pull request
7. Sur Github : accepter le pull request (auto-merging)
8. Aviser les autres membres de l'ajout

8. Aviser les autres membres de l'ajout

```
> git status    # Pour s'assurer que le dépot local est synchronisé

# Si local en retard
> git pull origin master    # Recupère les données distantes
# Si local en avance
> git push origin master    # Envoie les données vers origin

# Récupérer les modifications de upstream
> git pull upstream master   # Si présent, corriger les conflits
# Faire les tests (important!)

# mettre à jour son remote
> git push origin master

# Sur github, faire le pull-request et aviser les membres de l'ajout
```

RÉSUMÉ VISUEL