DIGITAL SEEDER

La baguette du ma**GIT**ien

Une aventure racontée par

Sébastien FLEURY

sebastien@digitalseeder.com

Il était une fois un artiste

Partir de rien
Imaginer et inventer
Ecrire et partager

Une œuvre collaborative

Faciliter les échanges
Historiser les avancements
Gérer les conflits

Historique de l’écosystème

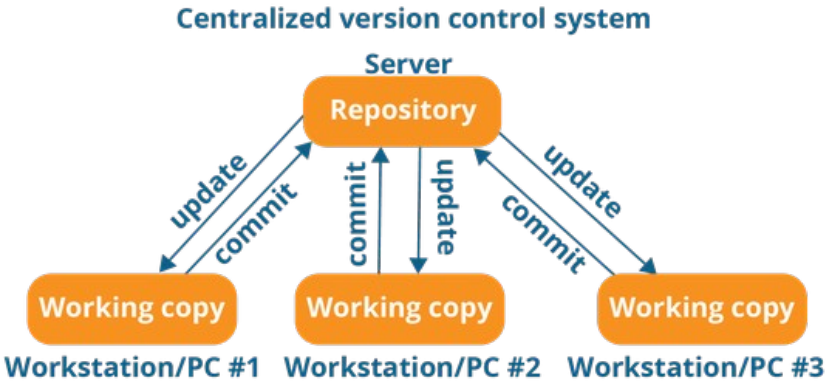
Systèmes centralisés (CVCS)

- CVS (Concurrent Versioning System, vieillissant)
- SVN (Subversion)

Systèmes décentralisés (DVCS)

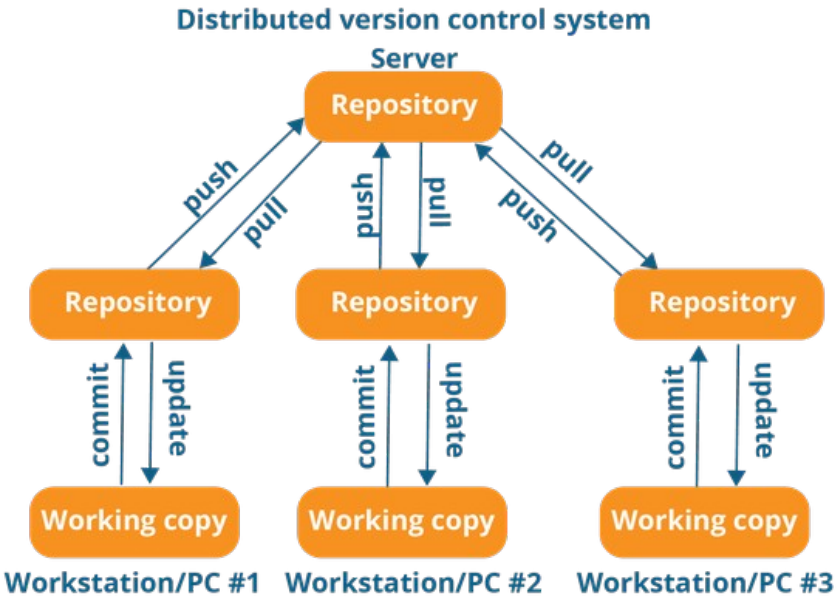
- GIT
- Mercurial (Hg)
- Bazaar (bzr)

Systèmes centralisés



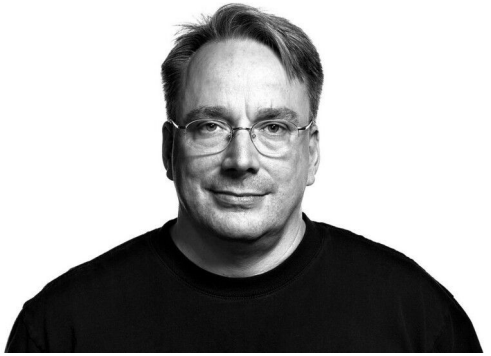
- Architecture client-serveur
- Un seul dépôt, le serveur central contient toutes les données
- Chaque commit une revision
- Récuperer les révisions avant de commiter
- Perte de temps et risques de conflits

Systèmes décentralisés



- Chaque développeur a un dépôt sur son poste
- Les commits n'impactent pas les autres collaborateurs
- On pousse sur un ou plusieurs dépôt distants
- On se partage un graphe

Git



- Créé en avril 2005 par Linus Torvald
- Objectif : gérer le workflow d’intégration des patches du noyau Linux
- Remplacer BitKeeper

How widely used is Git?

Adoption

Name	2015	2018
Git	**69.3%**	87.2%
Subversion	36.9%	16.1%
TFVC	12.2%	10.9%
Mercurial	7.9%	3.6%

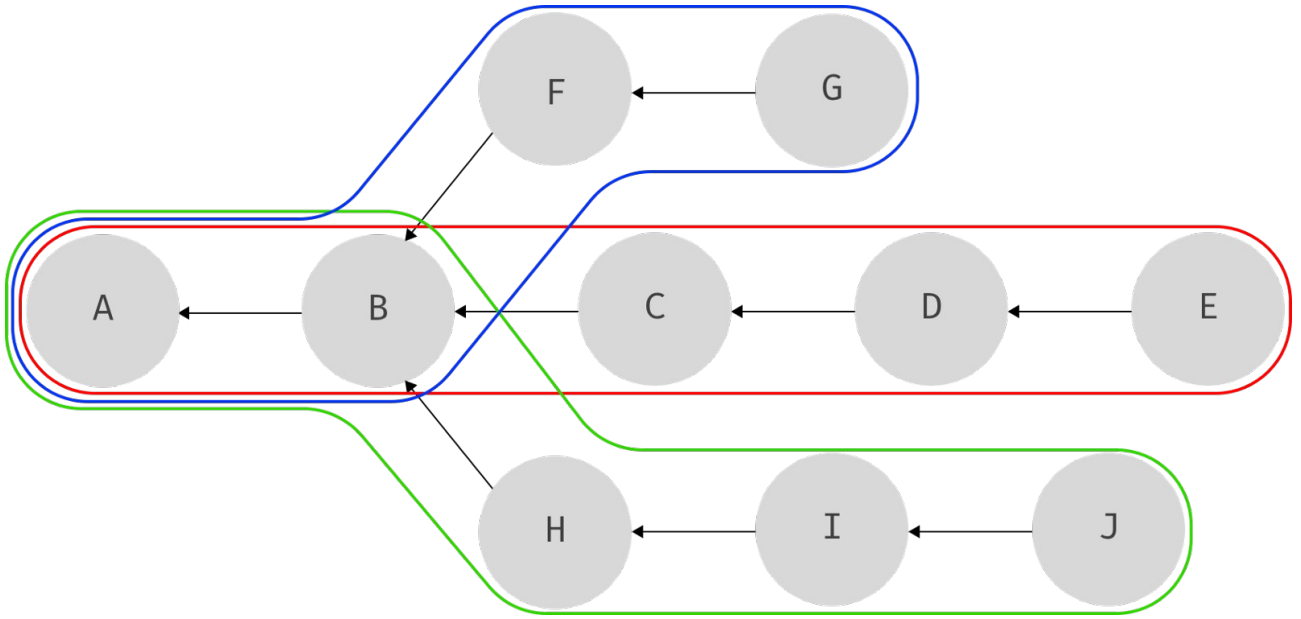
5 autres lignes

https://en.wikipedia.org › wiki › Git ▾

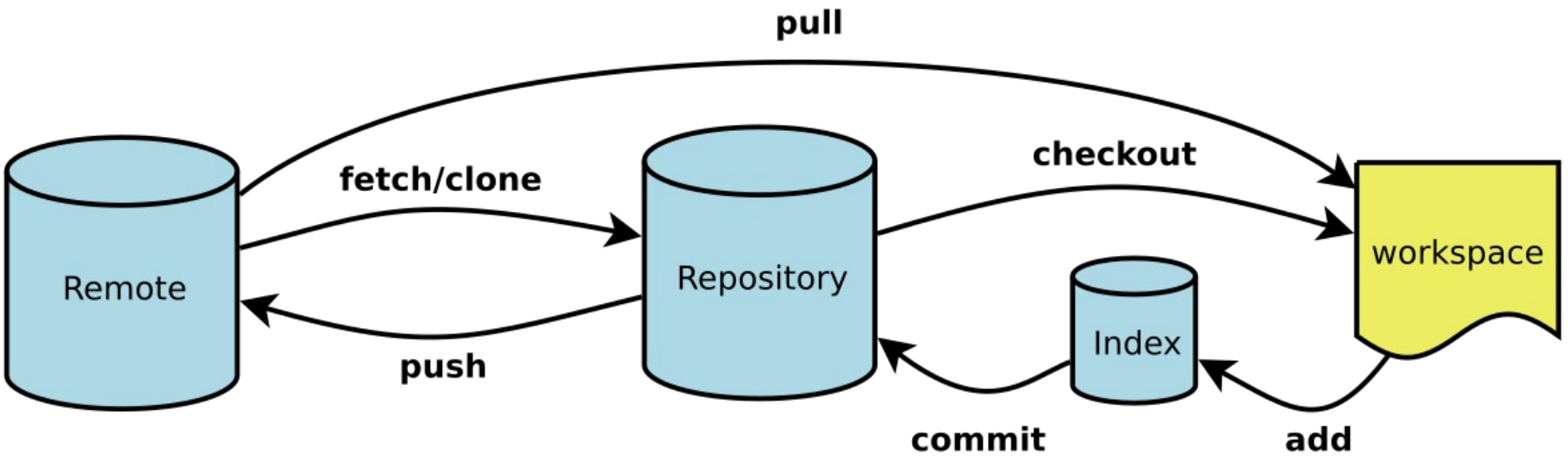
Un graphe ~~à 6 clics~~ acyclique orienté

En mathématiques, et plus précisément en théorie des graphes, un graphe est une structure composée d'objets dans laquelle certaines paires d'objets sont en relation. Les objets correspondent à des abstractions mathématiques et sont appelés sommets (ou nœuds ou points), et les relations entre sommets sont des arêtes (ou liens ou lignes)1. On distingue les graphes non orientés, où les arêtes relient deux sommets de manière symétrique, et les graphes orientés, où les arêtes, alors appelées flèches, relient deux sommets de manière asymétrique.

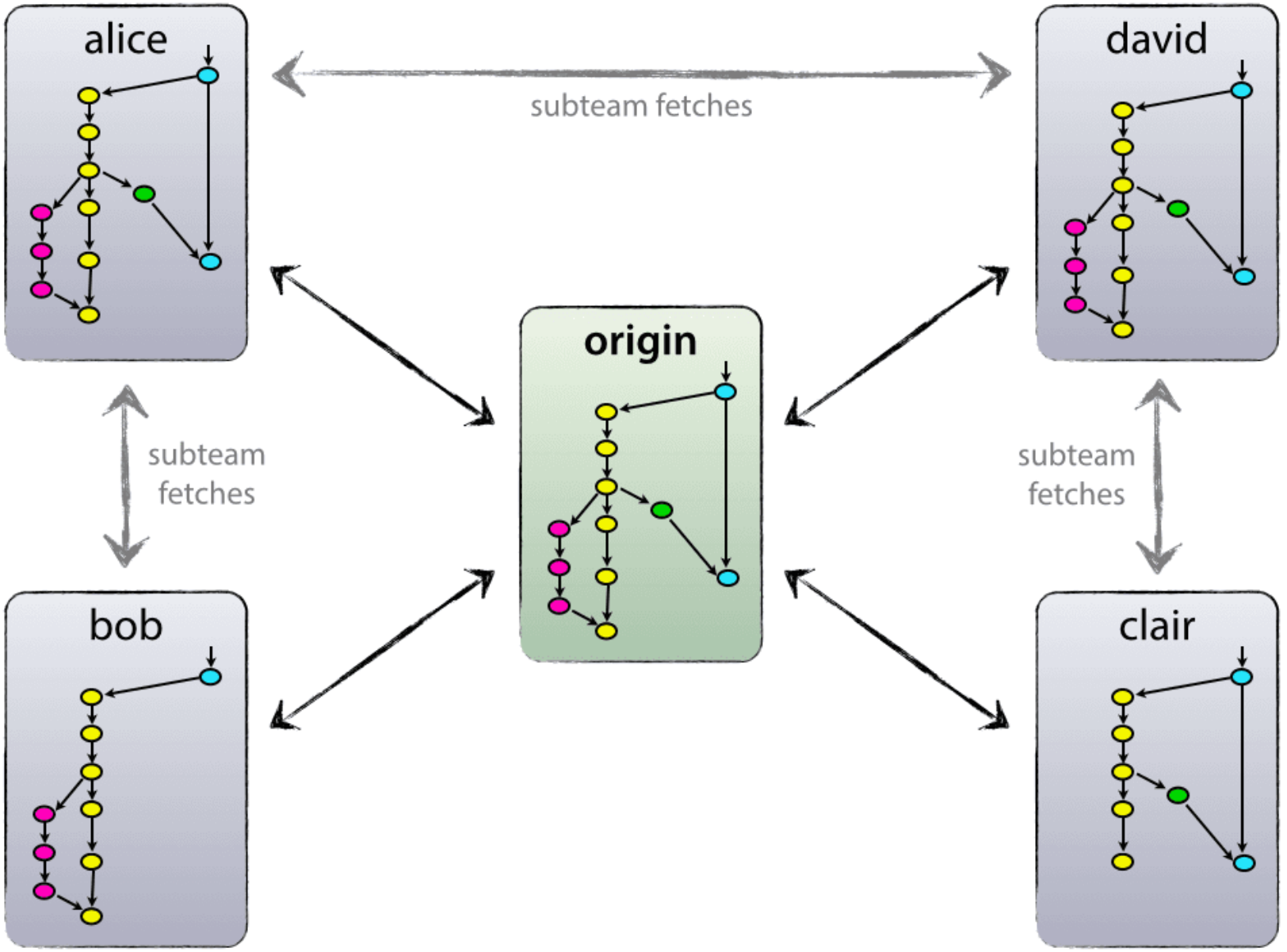
Wikipedia



Structure générale



Un graphe que l’on se partage



Écrivons ensemble une belle histoire

```
git init
```

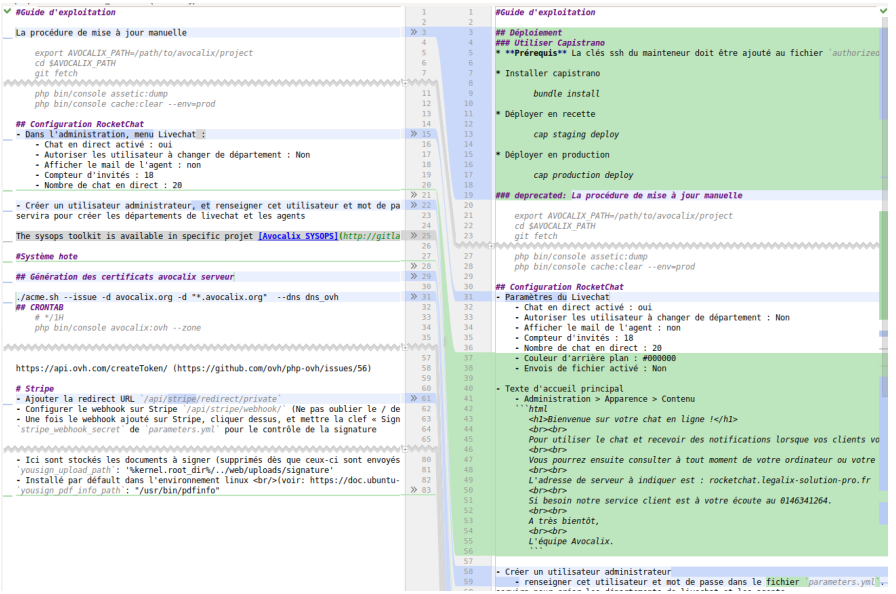
```
echo "# Formation Git" > README.md
```

```
git add -p README.md
```

```
git commit -m "Ici commence une nouvelle aventure"
```

Git simple gestionnaire de fichier ?

- Chaque commit est un diff avec l’état précédent
- Identifiés par une somme de contrôle SHA-1
- S’échange en pair à pair

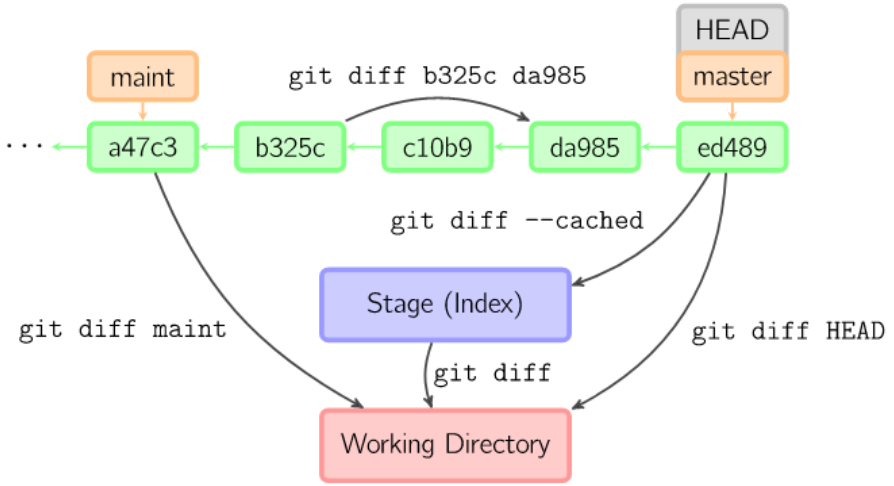


Les pointeurs

- HEAD, master
- Les branches (git checkout -b ma_branche)
- Naviguer dans les pointeurs avec ~
- Tous les pointeurs des dépots remote sont disponibles
  - Git branch –all

Archéologie 2.0

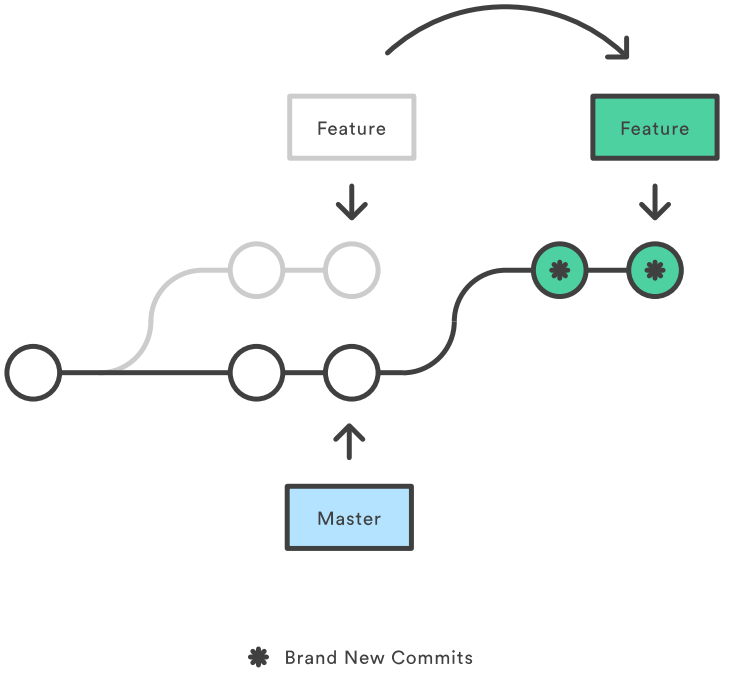
- Git log
  - Historique des commits
  - Chaque commit a un SHA1
- Git reflog
  - Historique des commandes Git
  - On peut revenir dans le temps
- Git diff
  - Etudier les changements entre deux étapes



Rejouer l’histoire

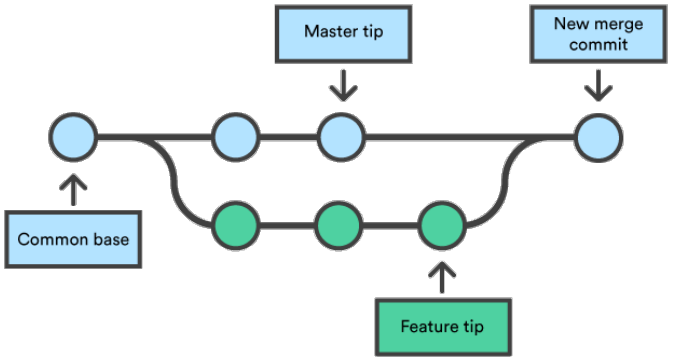
Git rebase

Détache les commits et les rejoue un par un



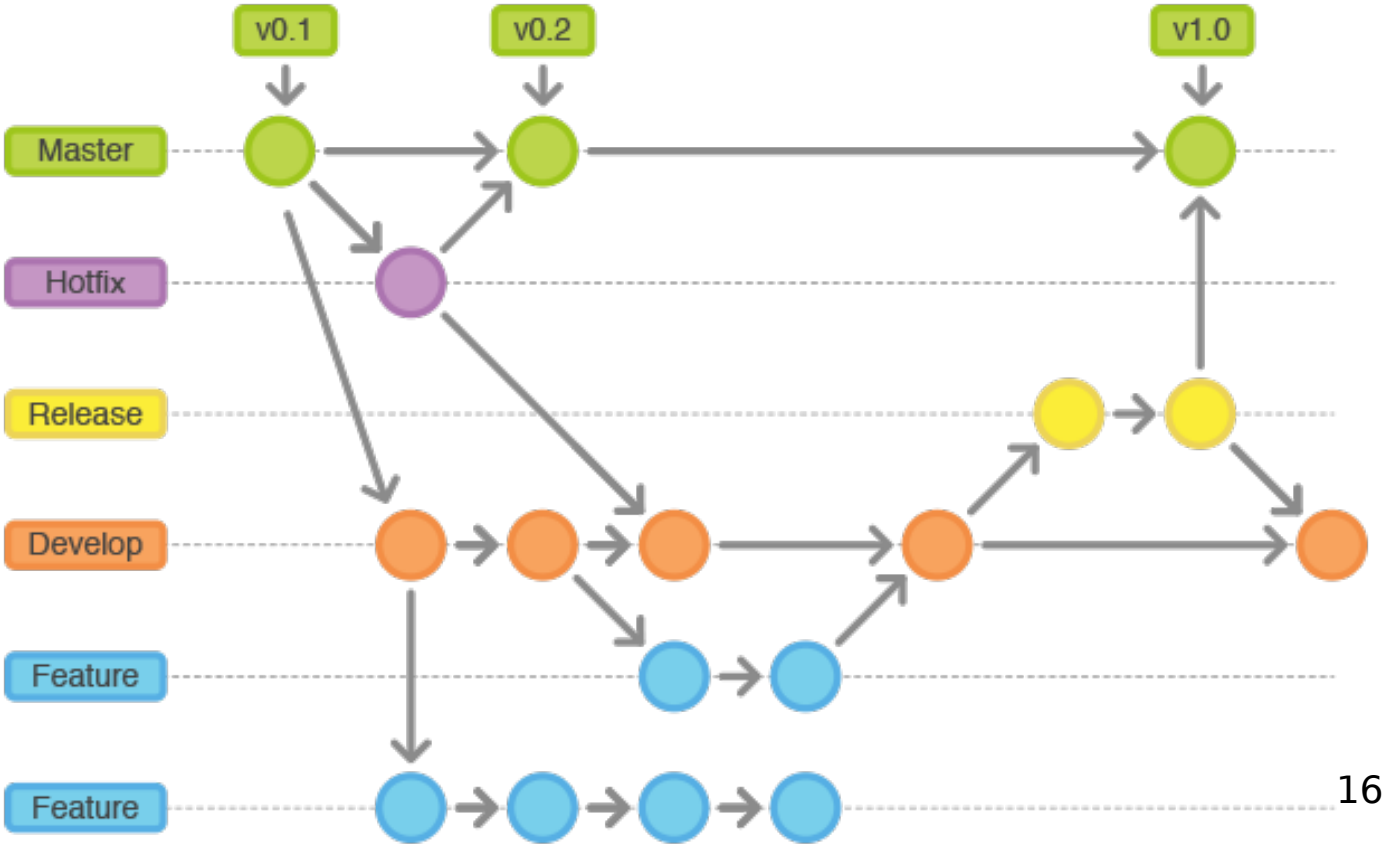
Valider les propositions

Git merge



Collaboration et flux de travail (Gitflow)

- Git branch
- Git checkout -b, git merge
- Git tag



Du bout des doigts

Consulter l'état du dépôt

- **git status**

Préparer son commit = ajouter les fichiers *unstaged* à la zone *staged*

- **git add -p**

Valider son commit = attacher l'état *staged* au commit pointé par HEAD

- **git commit -m 'un message explicite décrivant les modifications'**

Ouvrir une branche = créer un nouveau pointeur et déplacer HEAD dessus

- **git checkout -b 18_new_feature**

Récupérer le travail distant = mettre à jour son graphe et appliquer son travail local en tenant compte du travail récupéré (éviter *git pull*)

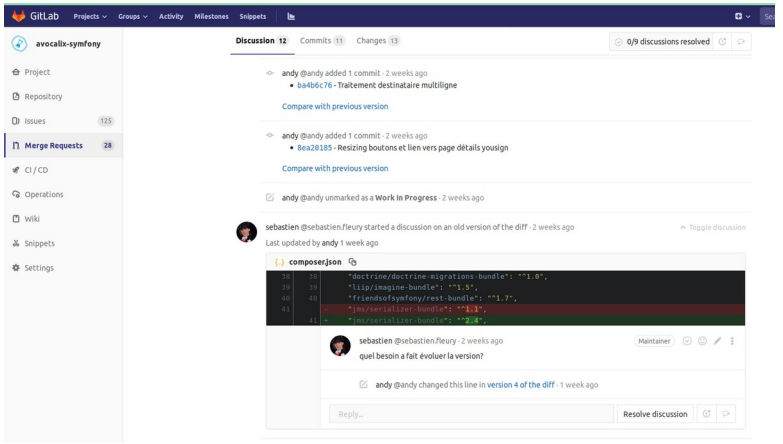
- **git fetch && git rebase**

Mettre son travail de coté = créer un commit temporaire et le placer dans la pile LIFO des commits de stash

- **git stash save 'un_nom_explicite_decrivant_ces_modifs'**

Soumettre un travail collaboratif

- Demander de fusionner une branche sur une autre = merge request / pull request
- Discussions directement sur le code
- Mettre a jour sa branche met à jour la PR



Références

- https://www.atlassian.com/git
- https://marklodato.github.io/visual-git-guide/
- https://onlywei.github.io/explain-git-with-d3
- https://www.tutorialdocs.com/article/git-basic-command-list.html

Quelques alias pratiques

Editer .gitconfig

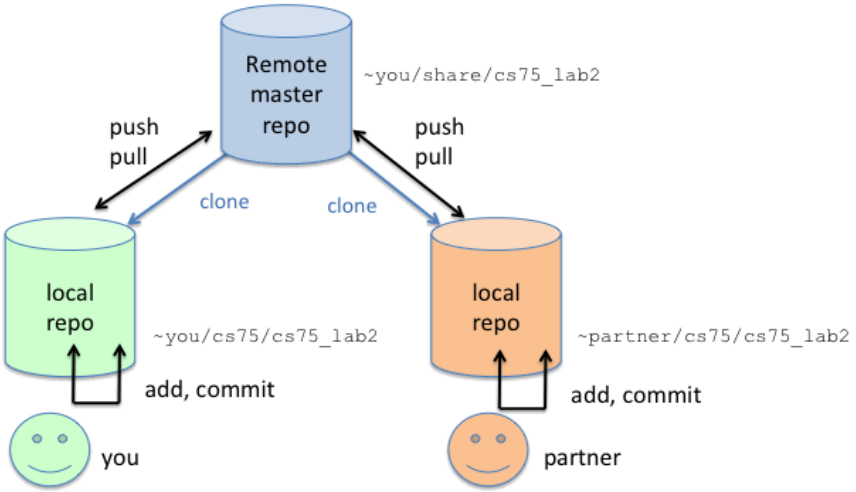
```
[alias]
st = status -sb
ci = commit
co = checkout
br = branch
branches = branch -a
glog = log --graph --abbrev-commit --date=relative
mergenff = merge --no-ff
#in = log master..origin/master
in = "!git remote update -p; git log ..@{u}"
out = log @{u}..
outall = log --branches --not --remotes=origin
lg = log --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit --color
di = diff
dic = diff --staged
id = show -s --pretty=format:'%h%d'
pullr = pull --rebase
who = "shortlog -ne --format='%h %s'"
rollback = reset --soft HEAD^
lc = log --pretty=oneline --abbrev-commit --graph --decorate ORIG_HEAD.. --stat --no-merges
latest = for-each-ref --sort=-committerdate --format='%(committerdate:short) %(refname:short)'
latestl = for-each-ref --sort=-committerdate refs/heads --format='%(committerdate:short) %(refname:short)'
authorship = "!git ls-files -z|xargs -0 -n1 -E'\n' -J {} git blame --date short -wCMcp '{}'| perl -pe 's/^.?\\((.?) +\\d{4}\\d{2}-\\d{2} +\\d+\\).*/\\1/'| sort | uniq -c | sort -rn"
 fix = "!_() { c=$(git rev-parse $1) && git commit --fixup $c && git diff-index --quiet HEAD; s=$?; [ $s != 0 ] && git stash; git -c core.editor=cat rebase -i --autosquash $c~ && [ $s != 0 ] && git stash pop; }; _"
 logone = log --pretty=format:'%C(auto)%m %h %Cgreen%ad %C(blue)%<(20)%aN %Creset%s %N %C(auto)%d' --left-right --cherry-pick --date=short
clear = branch -r | awk '{print $1}' | egrep -v -f /dev/fd/0 <(git branch -vv | grep origin) | awk '{print $1}' | xargs git branch -d
```

DIGITAL
SEEDeR

User Story 1

- **En tant que** développeur
- **Je souhaite** initialiser un projet Git
- **Afin de** travailler en collaboration avec mes camarades

Initialiser un projet



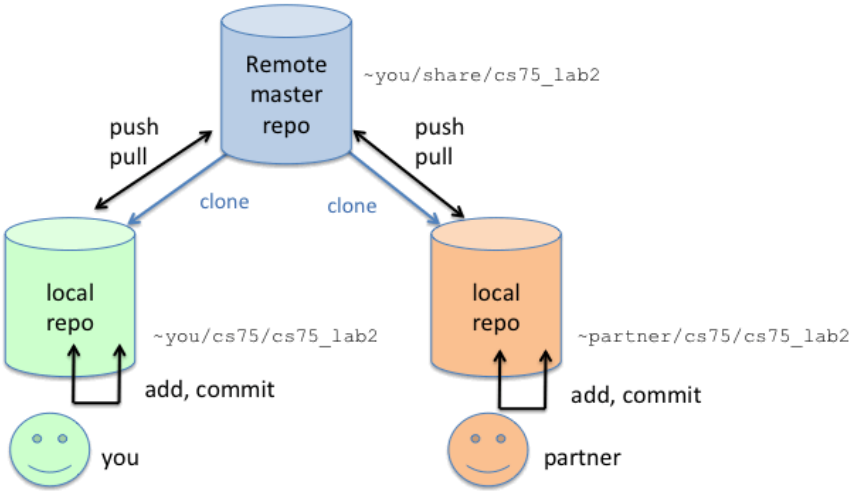
- Forker le projet
  - https://github.com/artmoni/processing-sample-project
- Cloner en local
  - Git clone https://github.com/artmoni/processing-sample-project
- Creer un commit local
  - Git add -p
  - Git commit -m «une description explicite »
  - Git push origin master

User Story 2

- **En tant que** développeur
- **Je souhaite** que chaque développeur travail dans une branche
- **Afin de** dissocier les fonctionnalités et limiter les conflits

User Story 2

- Créer une branche
  - Git checkout -b newFeature


Remote
master
repo
~you/share/cs75_lab2
push
pull
push
pull
clone
clone
local
repo
~you/cs75/cs75_lab2
local
repo
~partner/cs75/cs75_lab2
add, commit
add, commit
you
partner

Participer

- Créer un tableau de balles
- La couleur est définie aléatoirement à la création d’une balle
- Les balles diminuent de taille dans le temps
- Appuyer sur la touche « c » change le sens de parcours

Soumettre des évolutions

- Créer une branche
- Travailler sur sa branche
- Pousser sa branche sur son dépôt
- Créer une merge request vers le projet d’origine
- Récupérer l’avancement du projet d’origine
    - Git remote add artmoni https://github.com/artmoni/processing-sample-project.git
    - Git fetch
    - Git checkout master
    - Git rebase artmoni/master
    - Git push -f origin master