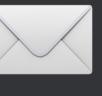


Git, DVCS & co



Bonjour

- Philippe Charrière
-  Clever Cloud <http://www.clever-cloud.com>
-  philippe.charriere@clever-cloud.com
-  @k33g.org
- emoji addict

Plan

- Git en local
- Git côté serveur
- Git à plusieurs
- Intégration continue

GIT?

- Systèmes de gestion de versions (distribué)
- VCS: *Version Control System*
- SCM: *Source Content Management*

GIT?

- Garde une trace historisé de votre travail
- Machine à remonter le temps
- Crée par Linus Torvalds en 2005
- Open Source
- Majoritairement utilisé dans le monde, même si...

Git fait peur



- Mode commande (il existe des GUI)
- Beaucoup de possibilités (*beaucoup de commandes*)

Je ne connais que ~10 commandes

- Restez le plus simple possible (on ne va pas tout voir)
- Si cela devient compliqué, ce n'est pas normal (*gestion de projet?*)
- 😢 Il y aura toujours les vieux projets...

C'est parti!

Paramétrer Git

```
git config --global user.name "busterbunny69"  
git config --global user.email "buster.bunny.69@gmail.com"  
git config --global credential.helper "cache --timeout=3600"
```

Check

```
git config --list
```

```
# ou git config user.name ...
```



```
user.name=busterbunny69
```

```
user.email=buster.bunny.69@gmail.com
```

```
credential.helper=cache --timeout=3600
```

Créez un projet: git init

`mkdir demo # en fait vous l'appelez comme vous voulez`

`cd demo`

`git init` ➡

`git init` permet de transformer votre dossier en **repository (local)**

git init

```
.git/  
| -- branches  
| -- config  
| -- description  
| -- HEAD  
| -- hooks  
| -- info  
|   '-- exclude  
| -- objects  
|   | -- info  
|   '-- pack  
'-- refs  
    | -- heads  
    '-- tags
```

1 repository == 3 arbres

- le **Working Directory** qui contient les fichiers en cours
- **Index** que l'on peut voir comme un "endroit de passage" / état temporaire / staging area
- **HEAD**: la dernière version (la plus récente) de votre projet

1ère utilisation

1ère utilisation

- Créez un fichier **README.md**
- Ajoutez du texte au format markdown 😯 ?



Ma vie, mon oeuvre

Hello A small blue and green globe icon representing Earth.

- Sauvegardez

git status

- Tapez `git status`



Untracked files:

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

`README.md`

nothing added to commit but untracked files present (use "`git add`" to track)

-  Vous êtes dans l'arbre **Working Directory**

git add <filename>

- Tapez `git add README.md` ou `git add *` ou `git add .`
- Tapez `git status`



Changes to be committed:

(use "`git rm --cached <file>...`" to unstage)

new file: README.md

-  Vous avez ajouté le fichier à l'arbre **Index (Stage)**

git commit -m "message"

- Tapez `git commit -m "add some text"` !



```
[master (root-commit) 56e0a78] add title  
 1 file changed, 2 insertions(+)  
 create mode 100644 README.md
```

- Vous avez ajouté le fichier à l'arbre **HEAD** (*version la plus récente*)
- Tapez `git status` pour voir où vous en êtes

Training

1- ajouter un sous-titre **## this is a sub-title**

- **git status (pour voir)**

- **git add .**

- **git status (pour voir)**

- **git commit -m "add a subtitle"**

2- ajouter du texte

- **git add .**

- **git commit -m "add some text"**

Training

⚠ Une bonne pratique est de faire qu'un commit corresponde à un tout/un ensemble logique (ex: une fonctionnalité)

Time machine - historique

- Tapez `git log`

```
commit 79eabfa3850804d1e7db24edbe8e13798d5c8aaf
```

```
Author: busterbunny69 <buster.bunny.69@gmail.com>
```

```
Date: Sun Aug 20 06:36:15 2017 +0000
```

add a subtitle

```
commit 56e0a78dd8f156c5eef7b71c0497ea619739d92d
```

```
Author: busterbunny69 <buster.bunny.69@gmail.com>
```

```
Date: Sun Aug 20 06:27:24 2017 +0000
```

add title

Time machine - historique

- Tapez `git log`

`commit 79eabfa3850804d1e7db24edbe8e13798d5c8aaf`

`Author: busterbunny69 <buster.bunny.69@gmail.com>`

`Date: Sun Aug 20 06:36:15 2017 +0000`

`add a subtitle`

`commit 56e0a78dd8f156c5eef7b71c0497ea619739d92d ➤ id de commit`

`Author: busterbunny69 <buster.bunny.69@gmail.com>`

`Date: Sun Aug 20 06:27:24 2017 +0000`

`add title`

Time machine > past

- Tapez:

```
git checkout 56e0a78dd8f156c5eef7b71c0497ea619739d92d
```

(sélectionnez l'id de commit le plus ancien)

Time machine > past

Note: checking out '[56e0a78dd8f156c5eef7b71c0497ea619739d92d](#)'.

You are **in 'detached HEAD' state**. You can look around, make experimental changes and commit them, and you can discard any commits you make **in this state without impacting any branches by performing another checkout**.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using -b with the **checkout command** again. Example:

```
git checkout -b <new-branch-name>
```

```
HEAD is now at 56e0a78... add title
```

Time machine > past

- Cette commande vous place dans un état appelé **detached HEAD**.
- Vous êtes revenu en arrière mais comme spectateur.
- Vous voyez le projet comme qu'il était à un instant **t**.

 allez voir le contenu de votre fichier

Revenir dans le présent

- Pour revenir dans le présent:
 - `git checkout *` (*uniquement si vous avez fait des modifications*)
 - `git checkout master`
- `master ? ? ?` ➔ c'est une "branche"

Plusieurs versions du même projet

"Oh, j'ai une idée, mais je ne voudrais pas tout casser"

- ajouter un chapitre / paragraphe
- ajouter une fonctionnalité
- ...

Mais sans casser mon travail en cours (je veux expérimenter)

Les branches



Créer une/des branche(s)

- Tapez: `git checkout -b wip-introduction`



Switched to a new branch 'wip-introduction'

- Tapez: `git branch`



`master`

* `wip-introduction` ➔ vous êtes ici

Revenir sur master

- Tapez: `git checkout master`
- Puis tapez: `git branch`



* **master** ➡ vous êtes ici
wip-introduction

Créez une autre branche

- Créez une branche `wip-chapter-1`
- Vérifiez
- Puis positionnez vous sur `wip-introduction`

Ecrivez votre introduction

- Editez **README.md** (*vérifiez que vous êtes sur la bonne branche*)
- Ajoutez votre modification à l'arbre **Index**
- Ajoutez votre modification à l'arbre **HEAD**

Faites le point: git log

- Tapez `git log --oneline`



42fea5a intro ➔

6a756cf add some text

79eabfa add a subtitle

56e0a78 add title

Créez votre chapitre 1

- Allez sur la bonne branche
- Editez **README.md** et ajoutez un chapitre (*vérifiez que vous êtes sur la bonne branche*)
- Ajoutez votre modification à l'arbre **Index**
- Ajoutez votre modification à l'arbre **HEAD**
- Allez lister l'historique

Votre intro est ✨ : merge !

- Vous allez reporter votre travail sur le projet en cours (**sur master**)
- Il faut se positionner **sur master**
- Et fusionner tout ça 😜

Votre intro est ✨: merge !

```
git checkout master
```

```
git merge wip-introduction # merge de wip-introduction sur master
```

Vous allez avoir un message de ce type:



```
Updating 6a756cf..42fea5a
```

```
Fast-forward
```

```
 README.md | 5 +++++
```

```
1 file changed, 5 insertions(+)
```

Même chose pour le chapitre 1 😊

■

...

Même chose pour le chapitre 1 😊

```
git merge wip-chapter-1
```



Auto-merging README.md

CONFLICT (content): Merge conflict in README.md

Automatic merge failed; fix conflicts and then commit the result.

Gestion des conflits

Pas de panique

Allez éditer le fichier

- Vous allez voir en 1er entre <<<<<< HEAD et ===== le changement "en cours", le dernier changement effectué
 - Et celui à venir (qui génère le conflit) entre ===== et >>>>>
- wip-chapter-1
- Modifiez (faites le ménage en enlevant les marques)
 - Sauvegardez

```
git status  
git add .  
git commit -m "this is my first chapter"
```

Récap

Tapez `git log --graph --full-history --color --oneline`

```
* 6cdd489 this is my first chapter
```

```
| \
```

```
| * 10ad60e my first chapter
```

```
* | 42fea5a intro
```

```
| /
```

```
* 6a756cf add some text
```

```
* 79eabfa add a subtitle
```

```
* 56e0a78 add title
```

ou plus détaillé `git log --graph --full-history --color`

Récap (encore mieux)

- ou encore mieux `git log --graph --full-history --color --oneline --decorate`

```
*   6cdd489 (HEAD -> master) this is my first chapter
| \
| * 10ad60e (wip-chapter-1) my first chapter
* | 42fea5a (wip-introduction) intro
| /
* 6a756cf add some text
* 79eabfa add a subtitle
* 56e0a78 add title
```

2, 3 petits trucs

- `git diff wip-introduction master` avant un merge par ex
- supprimer les branches dont on n'a plus besoin:

```
git branch -d feature_x
```

- supprimer une branche non mergée:

```
git branch -D branch_to_delete
```

Travailler avec un serveur

Travailler avec un serveur

- partager
- communiquer -> issues
- travailler en équipe
- sauvegarder
- gestion de projet (affectations, milestones, ...)
- ...

Serveurs DVCS (git compliant)

- GitHub <https://github.com/> 😍
 - SaaS + on-premises
- GitLab <https://about.gitlab.com/>
 - SaaS + on-premises + OSS
- BitBucket (Atlassian) <https://bitbucket.org>
 - SaaS + on-premises

"mini" serveurs DVCS (git compliants)

- Gogs <https://gogs.io/> en Go
- GitBucket <https://gitbucket.github.io/> 😍
- ...

Créez un compte dans GitBucket

- <http://polytech-group1.cleverapps.io/>
- ou ! (uniquement groupe 2) <http://polytech-group2.cleverapps.io/>

1- Sign in (en haut à droite)

2- Create new account

3- utiliser le même username et mail que votre client git local

4- logguer vous (Bouton Sign in)

Créez un projet

Directement dans GitBucket:

- en haut à droite: cliquez sur +
- choisir *New repository*
- nommer le projet **my-book** | laissez en *Public*
- checker *Initialize this repository with a README*
- cliquer sur *Create repository*

Ajoutez une branche & modifiez README.md

Directement dans GitBucket:

- Ajoutez: **wip-introduction** (*à partir de master*)
- Modifiez le README.md + commit
- Puis idem avec **wip-chapter-1** (*à partir de master*)

Mergez tout ça

Directement dans GitBucket:

- Allez dans la rubrique **Branches**
- Vous voyez 3 branches
 - **master**
 - **wip-introduction**
 - **wip-chapter-1**

Pull Request???

- Un super outil de communication
- Permet d'expliquer (et suivre) la "vie" d'une feature
- Permet d'échanger
- Permet de faire des revues de code
- Evolution du code (diff)
-  C'est mieux de faire la PR en début de code
- ...

Créez une Pull Request

- Créez une PR à partir de **wip-introduction**
- Commentez un peu votre PR
- Qualifiez votre PR (labels, ...)
- Ajouter du code, retournez dans la PR
- Et enfin mergez
- Allez vérifiez sur **master** la prise en compte

Créez une 2ème Pull Request

- Créez une PR à partir de `wip-chapter-1`
- Faites comme pour la précédente
- ... 

Résoudre le conflit

This branch has conflicts that must be resolved

Use the **command** line to resolve conflicts before continuing.

- Chez GitHub vous pourriez le faire directement dans l'IHM
- Dans notre cas il va falloir cloner le projet en local
- Fixer le conflit
- Pousser la modification

Cloner le projet

- Récupérer l'url pour cloner le projet (dans l'IHM)
 - ex: `http://polytech-group1.cleverapps.io/git/k33g/my-book.git`
 - ou **-group2**
- sur votre pose:

```
git clone http://polytech-group1.cleverapps.io/git/k33g/my-book.git
```

Modifier le code

```
cd my-book  
git branch # seulement master  
git branch -a  
  
* master  
  remotes/origin/HEAD -> origin/master  
  remotes/origin/master  
  remotes/origin/wip-chapter-1  
  remotes/origin/wip-introduction
```

Mergez wip-chapter-1 sur master

git checkout master # pour être bien sûr

git merge wip-chapter-1



Auto-merging README.md

CONFLICT (content): Merge conflict in README.md

Automatic merge failed; fix conflicts and then commit the result.

Editez et corrigez le README.md

my-book

=====

<<<<< HEAD

Introduction

...

=====

Chapitre 1

ma vie mon oeuvre

>>>>> wip-chapter-1

Préparez vous à publier

```
git add .
```

```
git commit -m "I added the chapter one"
```

```
git push origin master # je "pousse" ma branche master vers l'extérieur
```



```
Username for 'http://polytech-demo.cleverapps.io': k33g
```

```
Password for 'http://k33g@polytech-demo.cleverapps.io':
```

```
# ...
```

Préparez vous à publier



```
fatal: Authentication failed for 'http://polytech-demo.cleverapps.io/git/k33g/my-book.git/'
```

Il vous faut une clé SSH

```
# Générer une clé SSH  
ssh-keygen -t rsa -C "buster.bunny@gmail.com"  
  
# Afficher la clé SSH  
cat ~/.ssh/id_rsa.pub  
👉  
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQDT11SDweP4e4kmNE16WMcXZ+c/2NN07gM  
+t9Y5J52f1UBCrAqDGoLBgAP9nP6f68sXePwxF6b91aRCvFBG5y5q7BfdtttsExUjfTdC6IkAC9  
wFjp2layS/kC3bsBmzI3y4iJtmdxlpAb1lFb3IuQb3MQl9KdY2aNDRT0Dup8ZKh4FLORK9DxD  
Kmy2o3TsSwJFEHcpRrHcPWCGqfK3G40GHg2kWYFM6/ghRxrY7FrU4a+boHqVOWD5661m00EKKy8AUhYc  
Hpm1CCQrq15xKl1VEh7nFn95rrdcnkRhloPtfn2TI51zQ4kfGJTMc5meQySTpafNd8KFVS  
wKfZzBDrDVb buster.bunny@gmail.com
```

Il faut aller l'ajouter dans votre profil GitBucket

- Copiez la clé (*à partir de ssh-rsa*)
- Allez dans votre profil GitBucket
 - *Edit your profile*
 - *SSH Keys* -> créez une clé et coller le contenu dans la rubrique **Key**
- C'est la même chose chez GitHub, GitLab, BitBucket, ...

Git push again

```
git push origin master
```



```
Username for 'http://polytech-demo.cleverapps.io': k33g
Password for 'http://k33g@polytech-demo.cleverapps.io':
Counting objects: 3, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 339 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote: Updating references: 100% (1/1)
To http://polytech-demo.cleverapps.io/git/k33g/my-book.git
 b2872ba..ec4be96  master -> master
```

Retournez voir votre dernière PR

- Faites un refresh si nécessaire
- 😊 Il n'y a plus de conflit
- Update and merge
- Aller voir le **README.md sur master**
- ⚠ Ne pas oublier de faire un **git pull** pour mettre à jour votre

2, 3 trucs

- `.gitignore`
- `.gitkeep`

Références

- <http://rogerdudler.github.io/git-guide/>
- les emojis c'est important:
 - <https://gitmoji.carloscuesta.me/>
 - <https://www.webpagefx.com/tools/emoji-cheat-sheet/>

Maintenant, on bosse en équipe

Maintenant, on bosse en équipe

Groupe (Organisation) polytech

Projet (repository) hello-service

- <http://polytech-group1.cleverapps.io/polytech>
- ou <http://polytech-group2.cleverapps.io/polytech>

De mon côté

-  vous enregister dans le groupe

Cloner le projet

```
git clone http://polytech-group1.cleverapps.io/git/polytech/hello-service.git
```

⚠ OU

```
git clone http://polytech-group2.cleverapps.io/git/polytech/hello-service.git
```

Exercice 1

👉 Faire la doc de ce que vous allez coder

En local sur votre poste:

- créez une branche **wip-doc-<votre-nom-user>**
- créez un fichier **doc-<votre-nom-user>.md** (*avec un peu de contenu*)
- "commitez"
- "poussez" (votre branche) sur le serveur

Exercice 1

```
git checkout -b wip-doc-<votre-nom-user>
git branch # pour vérifier ...
# créez un fichier doc-<votre-nom-user>.md
# ajoutez un peu de texte
git add .
git commit -m "my file"
git push origin wip-doc-<votre-nom-user>
# retournez sur GitBucket
```

Exercice 2

- Ajouter encore du code dans votre document
- Mettez à jour votre PR
- Notifyez vos petits camarades de ce que vous faites 🤔 ?
- Mergez votre PR sur **master**
- ⚠️ Mettez à jour votre repo local

```
git checkout master
```

```
git pull
```

Exercice 3: ajoutez une feature au service

- créer une branche locale: wip-hello-<votre-nom-user>
- créer un fichier js <votre-nom-user>_service.js

```
function restAPI(service) {  
  service.get({uri: "/api/k33g", f:(request, response) => {  
    response.sendJson({message:"hello I'm k33g"})  
  }})  
}  
  
module.exports = {  
  restAPI: restAPI  
}
```

Exercice 3: ajoutez une feature au service

- `git add . | git commit | git push`
- Allez sur GitBucket et créez la PR à partir de votre feature branch
-  ne mergez pas!!!
- Il va falloir modifier `index.js`

Exercice 3: ajoutez une feature au service

Maintenant vous devez intégrer votre service dans `index.js`

```
const {Service} = require('./pico')

const k33gAPI = require('./k33g_service').restAPI ➡️ 🙌

let helloService = new Service({})
let port = process.env.PORT || 9090;

helloService.get({uri:`/api/hello`, f: (request, response) => {
  response.sendJson({message: "Hello 🌎", from:"pico"})
}})

k33gAPI(helloService) ➡️ 🙌

//...
```

Exercice 3: ajoutez une feature au service

- `git add . | git commit | git push`
- Mergez, enfin essayez ...

Donc ! toujours synchroniser sa feature branch avec master

```
git checkout master  
git pull  
git checkout wip-hello-buster  
git merge master  
-> si conflits -> corriger conflits  
git add . | git commit  
git push origin your-feature-branch
```



Evitez de bosser à plusieurs sur la même branche



Faites de petites features

Workflows

- Importants
- Adhésion (et respect) de l'équipe
- Un bon workflow peut vous simplifier la vie
- ... surtout si il est simple 😊

Workflows

- Git Flow (*très utilisé*) <https://danielkummer.github.io/git-flow-cheatsheet/>
- GitHub Flow (*très simple*) <https://guides.github.com/introduction/flow/>
- Votre Flow - adaptez selon votre besoin

GitHub Flow

The GitHub Flow - Experimentation without risk
Open a Pull Request
discuss & conclude

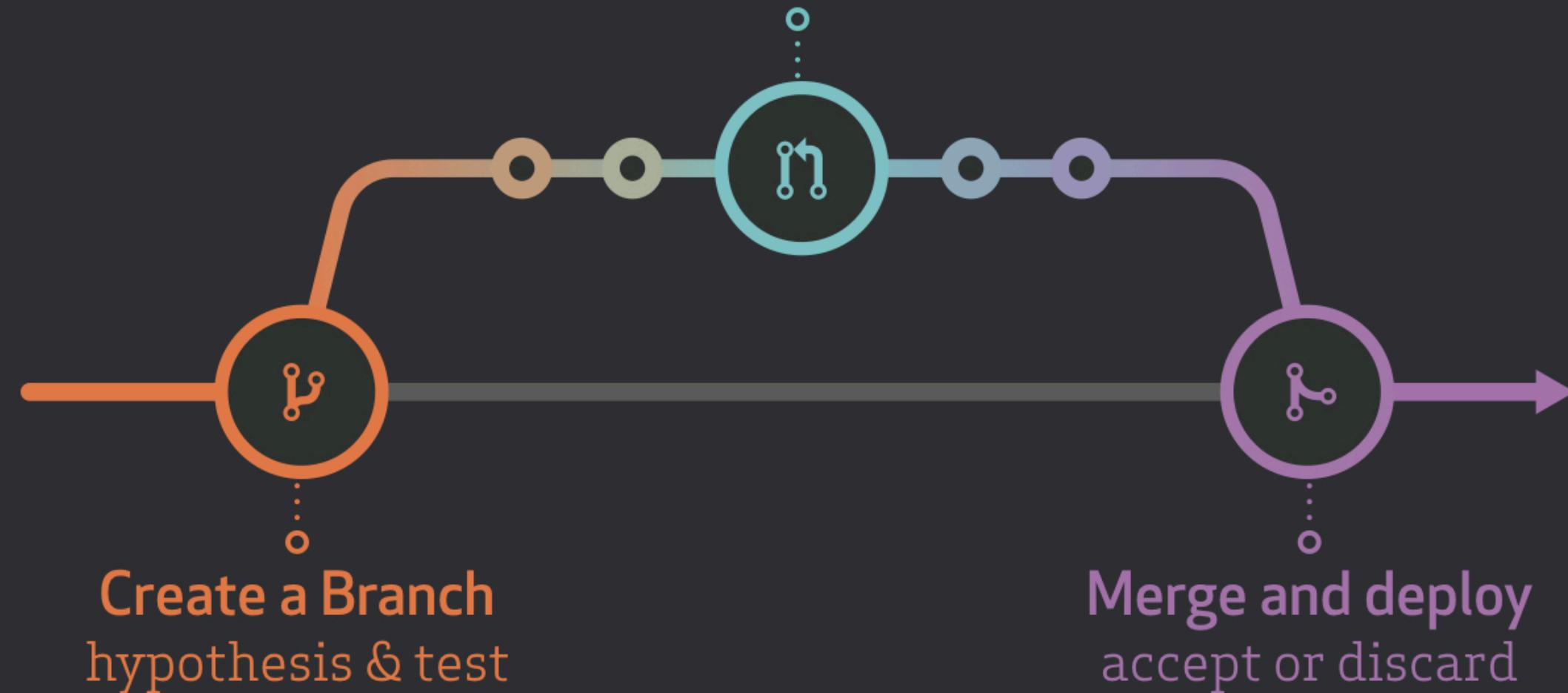


The GitHub Flow - Experimentation without risk

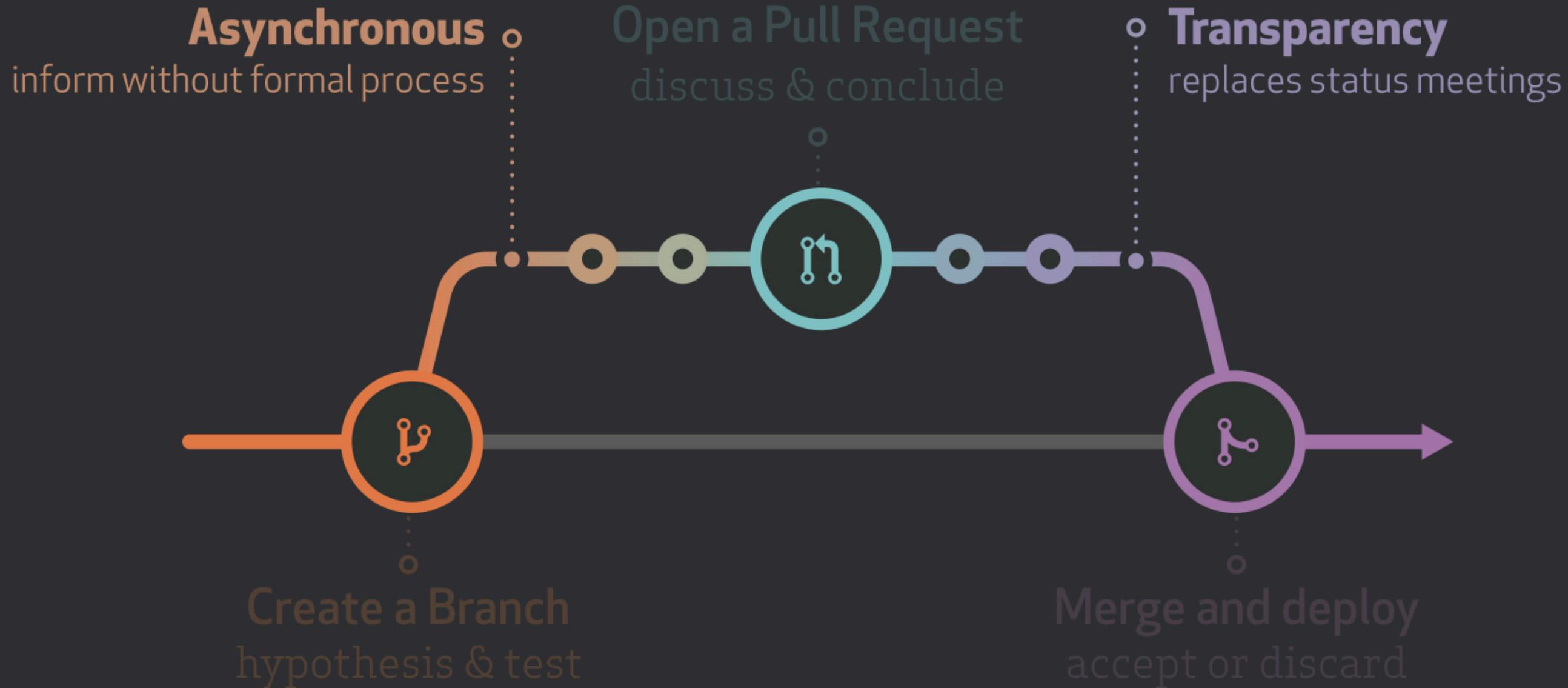


Open a Pull Request

discuss & conclude



The GitHub Flow - unlock team velocity



Autre modèle pour travailler à plusieurs: le Fork



- très utilisé dans le monde open-source
- <https://help.github.com/articles/working-with-forks/>

Comment gérer des versions et des fixes?

Le tag

- Un alias dont le rôle est de pointer sur un commit (plus facile à identifier)
- C'est aussi utile pour identifier une version



```
git tag -a v1.0.0 -m "my 1st version"
```

```
git push origin v1.0.0
```

```
git show v1.0.0 # soyez explicites dans le descriptif du tag
```

Le tag c'est top pour gérer les fixes

- 🐞 Sur une version précédente
- il faut faire un fix sur cette version

```
git checkout -b fix-oups v1.0.0 # créer une branche à partir du tag  
# faites vos modifs  
git add .  
git commit -m "my fix"  
git push origin fix-oups
```

Le tag c'est top pour gérer les fixes

- 📦 On reporte le fix dans la version en cours

```
git checkout master # si on décide de reporter le fix  
git merge fix-oups  
git push origin master  
git tag -a v1.0.1 -m "my fix"  
git push origin v1.0.1  
git show v1.0.1
```

(on aurait pu faire une PR)

**Vous pouvez essayer
sur votre projet**

A savoir

- Grâce à **Git** on peut synchroniser plusieurs serveurs
- Par exemple, vous gérez votre projet sur un serveur
- Mais vous souhaitez le publier sur 2 autres

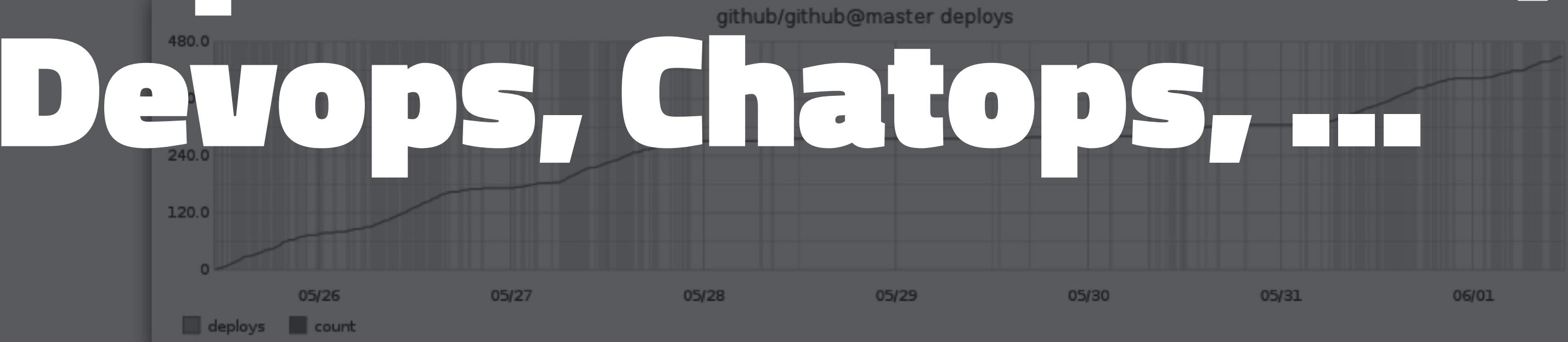
```
# dans mon repository
git remote add polytech1 http://polytech-group1.cleverapps.io/git/polytech/hello-service.git
git remote add polytech2 http://polytech-group2.cleverapps.io/git/polytech/hello-service.git
```

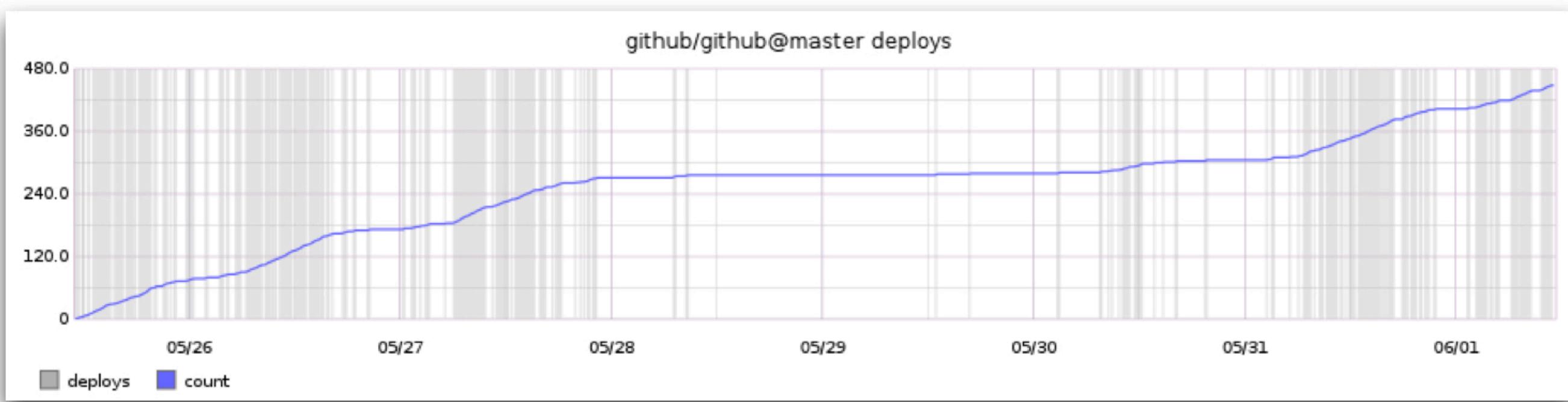


```
git push polytech1 master
git push polytech2 master
```

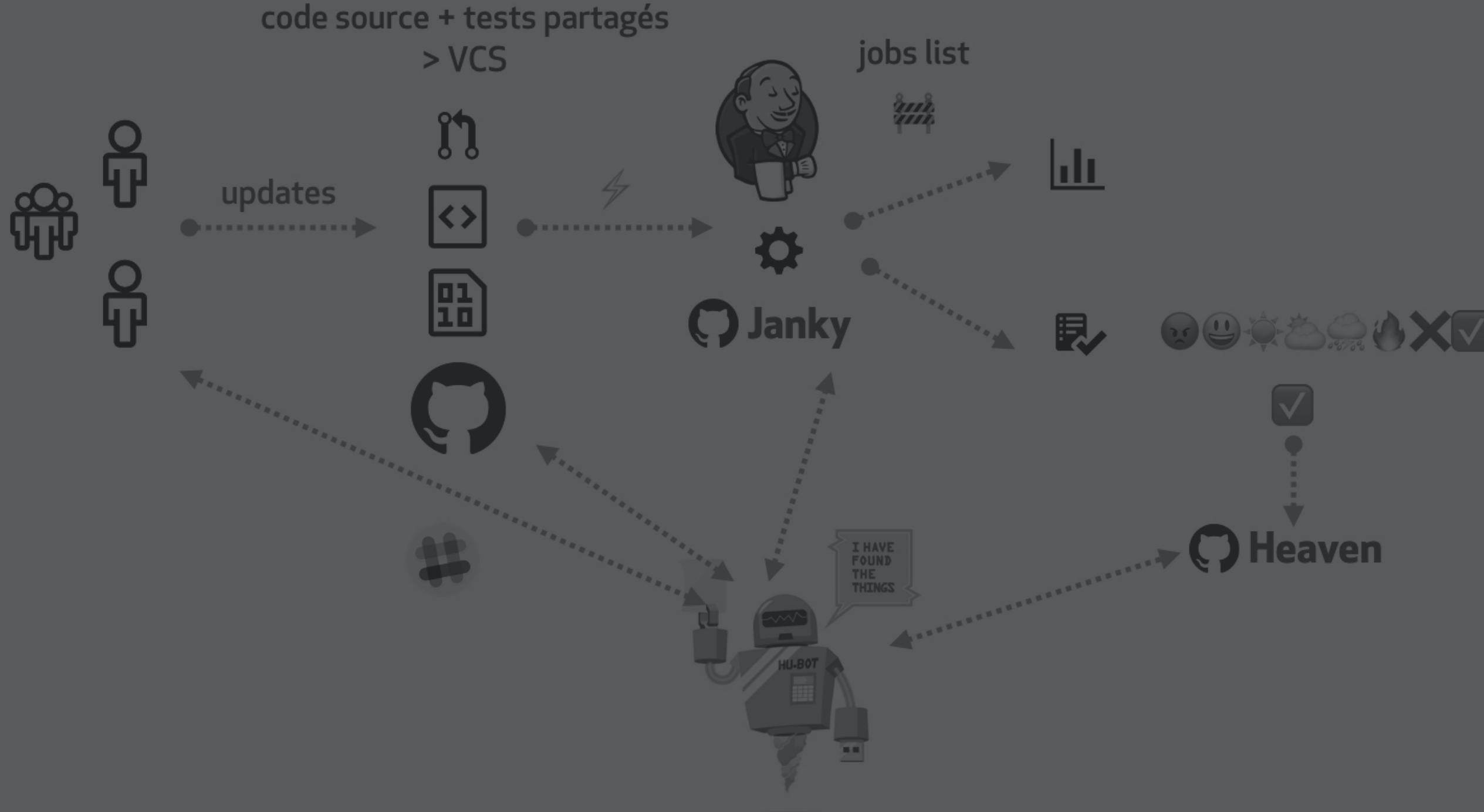
Intégration Continue, Déploiement continu,

Devops, Chatops, ...

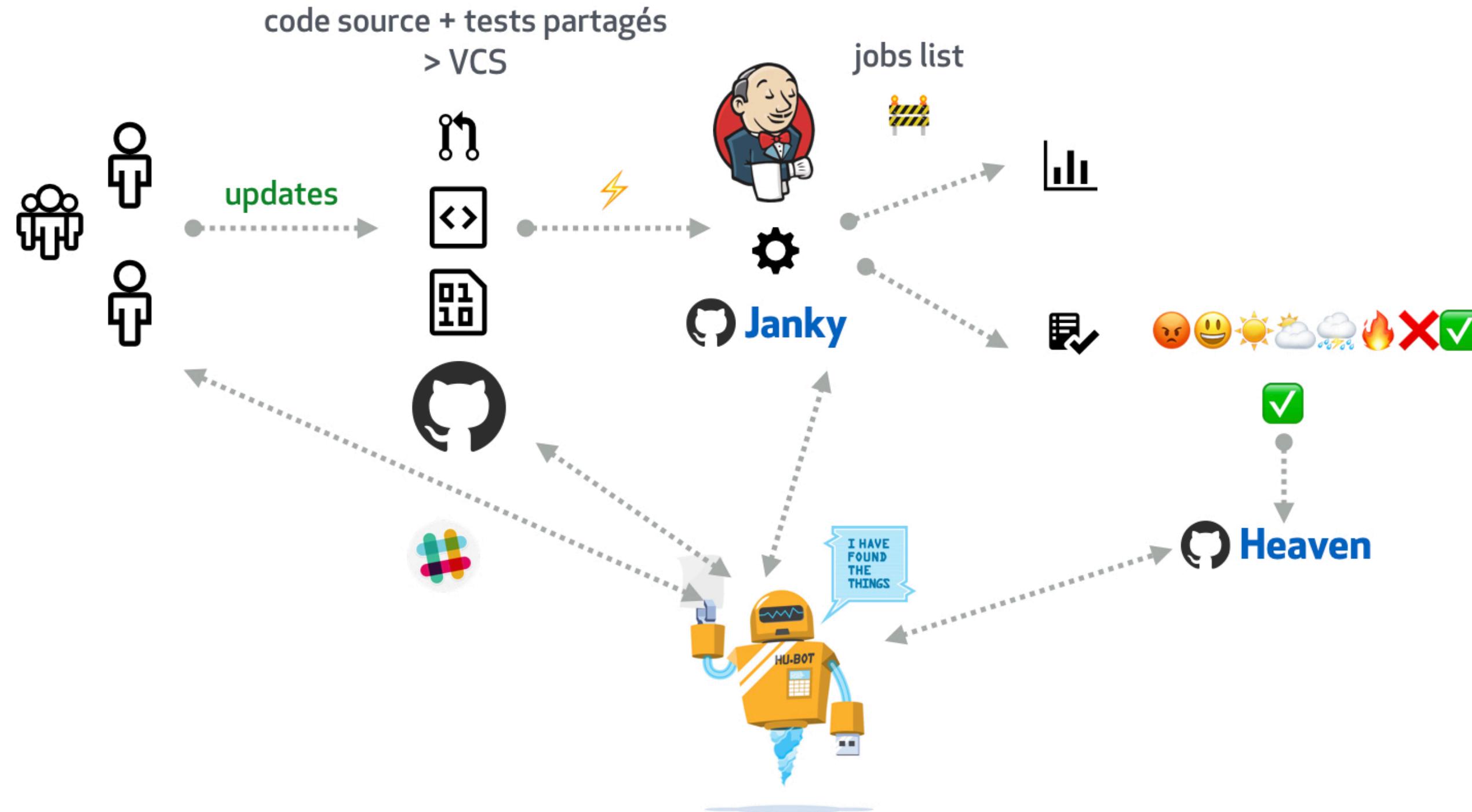




Il faut des outils



La chaîne | Chatops + **Déploiement** continu



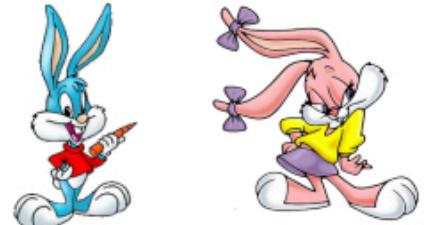
Flow



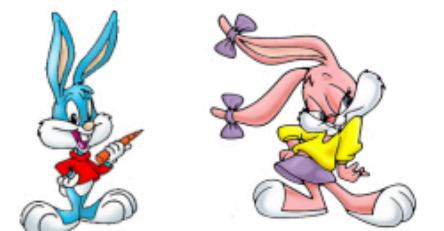
Issue

@mention

WEB HOOKS



USERS/BOTS

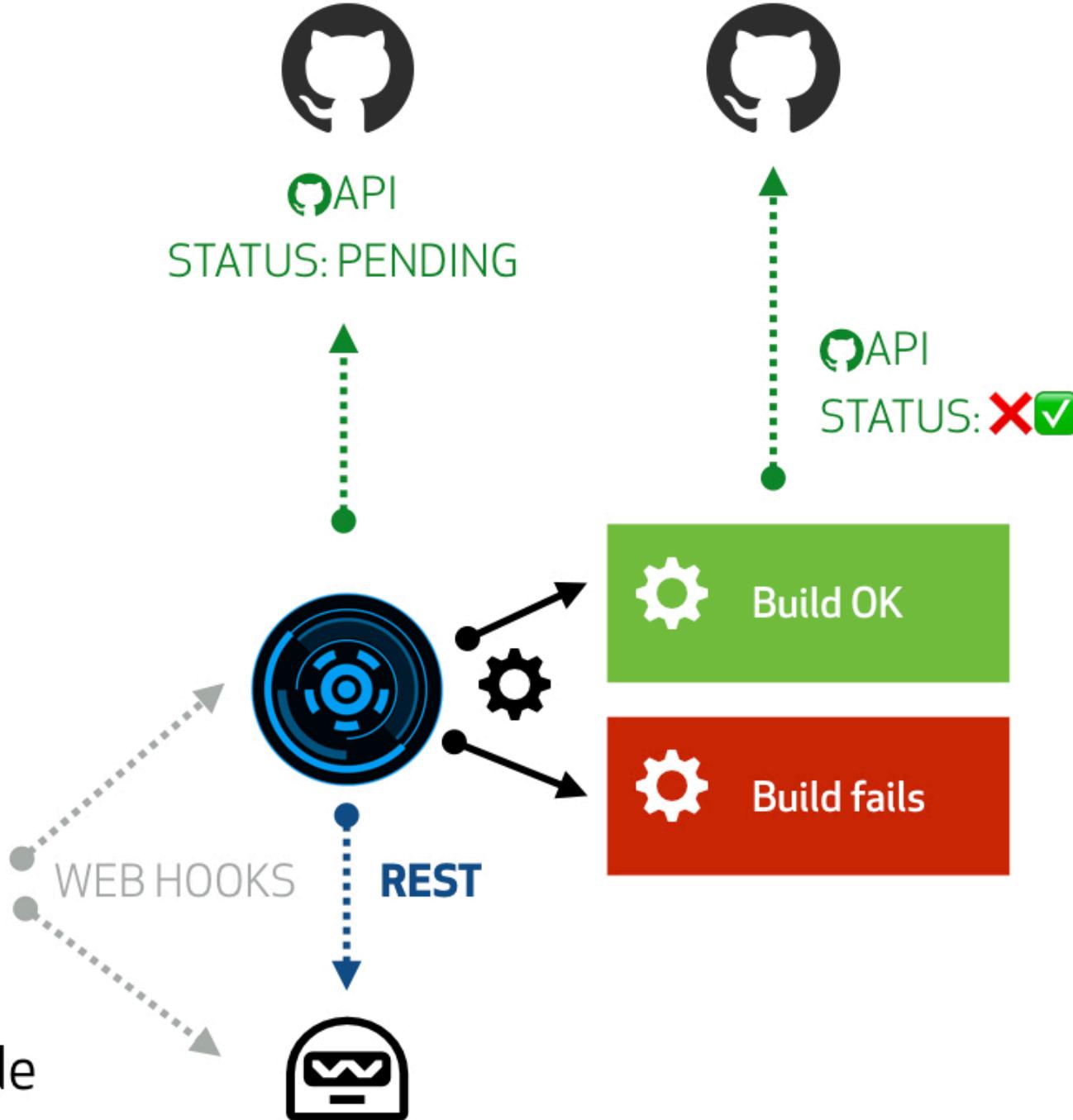


Create a branch

Add commits

Open a Pull Request

👋 hey this is my code



Discuss and review code

@mention

Add commits

Publish

Build OK

Merge

Petite Démo

- DVCS: <http://gitbucket-app-devops-demo.cleverapps.io/>
- CI: <http://jenkins-app-devops-demo.cleverapps.io/>
- Chat <http://letschat-app-devops-demo.cleverapps.io/>