

# Environnement de travail collaboratif


---

ESGI - Semaine du 04/11/19

Christophe Villegier  
Vincent Monjaret  
Lucas Moisan

# Introduction

---

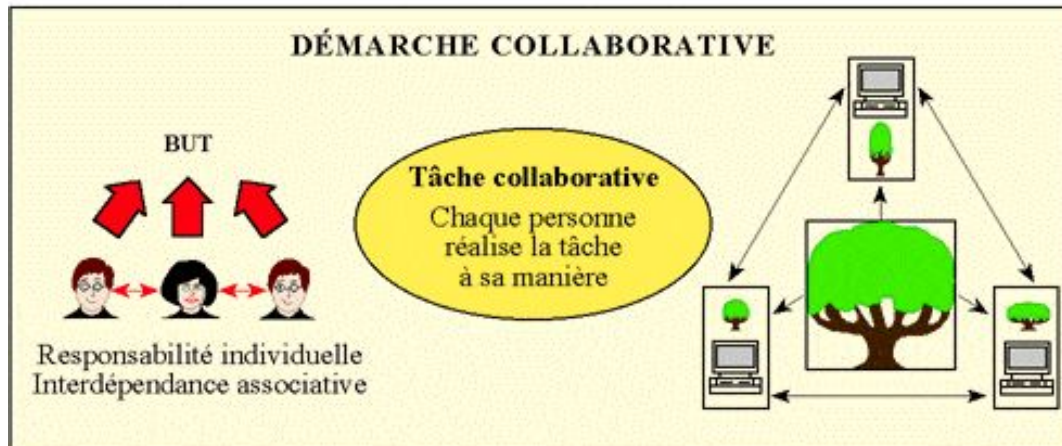
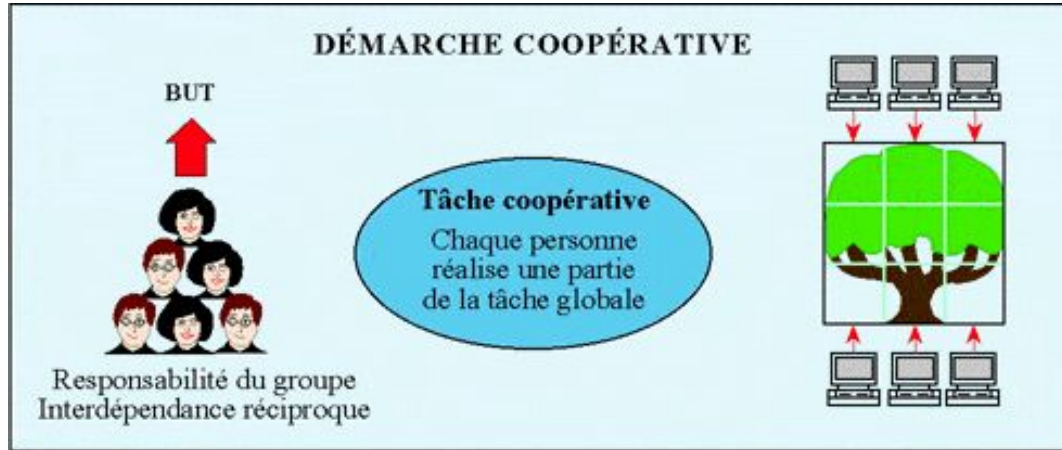
A photograph of three mountaineers ascending a steep, snow-covered mountain slope. The climbers are wearing full mountaineering gear, including helmets, backpacks, and ice axes. They are spaced out along a path, leaving tracks in the snow. The sky is a deep, clear blue, and the mountain's peak is visible in the background. The text 'Travail collaboratif : pourquoi est-ce indispensable ?' is overlaid in the center of the image.

**Travail collaboratif :  
pourquoi est-ce  
indispensable ?**

# Travail collaboratif : pourquoi est-ce indispensable ?

- ★ **79 %** des échanges se font par **mail** au sein des organisations. L'usage d'une plateforme digitale permet de désengorger la boîte mail.
- ★ **37 %** des sondés **ne retrouvent pas les documents** qu'ils recherchent dans la base documentaire de l'organisation.
- ★ **58 %** des sondés **n'ont pas une totale liberté** de leurs outils informatiques, tant en termes de choix que d'usages.
- ★ **1/3** des collaborateurs déplore que **ses attentes ne soient pas prises en compte** dans les projets informatiques

# Travail collaboratif : pourquoi est-ce indispensable ?



# Travail collaboratif : pourquoi est-ce indispensable ?

*Les 3 principaux freins à l'adoption d'une plateforme collaborative :*

- ❖ La **résistance** au changement
- ❖ L'absence d'appui du **management**
  - ❖ Une organisation en **silos**

*Les 3 principaux facteurs déclencheurs de l'équipement d'une plateforme collaborative :*

- La recherche **d'efficacité**
- La transformation **digitale**
  - Le gain de **temps**

# Travail collaboratif : pourquoi est-ce indispensable ?

- **Mutualiser** les connaissances et compétences
- Mettre en avant la **transparence**
- Favoriser la **communisation** en s'assurant que chacun puisse **s'exprimer**
- Vive la **mobilité** : Gérer vos projets où que vous soyez

# Git

---





# Initialize a repository: **init** & **clone**

→ `git init`

Initialize a directory to make it able to use git

→ `git clone <url> [<directory>]`

Creates a directory and clones the entire repository inside it.

# Don't be anonymous

→ `git config --global user.name "Marty McFly"`

→ `git config --global user.email "luke.skywalker@tatooine.xyz"`

# Getting on the stage

Staged modifications are the one that will be added in your next commit

→ `git status`

Shows the status of the modified files of the working directory

→ `git status`

On branch master

Changes to be committed:

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

modified: another\_file.txt

Changes not staged for commit:

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

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

deleted: a\_file.txt

Untracked files:

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

yet\_another\_file.txt

# Getting on the stage

→ `git add <file>`

Stages a file e.g. adds as it currently is to the next commit

→ `git add .`

Stages every modifications

→ `git add -p <file>`

Choose interactively which changes should be staged

→ `git reset [<file>]`

Unstage the modifications on a file or the entire repository, but keep them in the working directory

→ `git reset --hard [<file>]`

Unstage the modifications **and** cancel the modifications in the working directory

→ `git rm <file>`

Delete a file and stage its deletion

→ `git mv <path> <new-path>`

Move a file and stage its deletion

→ `git commit -m "A beautiful, concise and clear message"`

Add your staged content as a new commit

# Getting on the stage

```
→ echo "a first modification" > a_file.txt  
→ git add a_file.txt  
→ echo "a second modification" > a_file.txt  
→ git commit -m "I made great modifications"
```

What will be committed?

# A great tree with great branches

```
→ git branch  
    another-branch  
* i-really-love-branches  
  master
```

List branches

```
→ git branch <branch-name>
```

Creates a new branch

```
→ git checkout <branch-name>
```

Switch to another branch and check it out in your working directory

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

Creates a new branch and switch to it

# Learn your history

→ `git log`

```
commit 66f89dd03e4dd3a1182c2c0dc9ec5542f55b3ca4 (HEAD -> master)
Author: Lucas Moisan <lmoisan.ext@wedigital.garden>
Date:   Wed Oct 30 23:17:14 2019 +0100
```

Fixed a bug (that feature was not that awesome.

```
commit efcf8fc5f2116b01e5cfdc3adc9231650e5602a0
Author: Lucas Moisan <lmoisan.ext@wedigital.garden>
Date:   Wed Oct 30 23:16:43 2019 +0100
```

Added a great feature.

```
commit 10f86f23deea7d603b9f4d3b27c0ed14961d6450
Author: Lucas Moisan <lmoisan.ext@wedigital.garden>
Date:   Wed Oct 30 23:16:18 2019 +0100
```

Initial commit.

# Learn your history

→ `git log ---oneline`

```
66f89dd (HEAD -> master) Fixed a bug (that feature was not that awesome.  
efcf8fc Added a great feature.  
10f86f2 Initial commit.
```

Displays commits on one line only

→ `git config --global alias.lg "log --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<an>%Cr<reset' --abbrev-commit"`

→ `git lg`

```
* f5a245a - (HEAD -> master) Merge branch 'another-branch' (42 seconds ago) <Lucas Moisan>  
|\n| * 7212921 - (another-branch) Finish a new feature. (73 seconds ago) <Lucas Moisan>  
| * acc2eda - Begin a new feature. (2 minutes ago) <Lucas Moisan>  
* | c8a3e01 - Fix a critical bug. (52 seconds ago) <Lucas Moisan>  
|/  
* 66f89dd - Fixed a bug (that feature was not that awesome. (6 minutes ago) <Lucas Moisan>  
* efcf8fc - Added a great feature. (7 minutes ago) <Lucas Moisan>  
* 10f86f2 - Initial commit. (7 minutes ago) <Lucas Moisan>
```

A great log alias



# Working with remotes

→ `git remote add [<remote-alias>] [<url>]`

Add a git url from which you will be able to pull / push updates

→ `git fetch [<remote-alias>]`

Fetch branches from a remote

→ `git push [<remote-alias>] [<branch>]`

Transmit your current local branch to a remote branch.

If you just added commits, you will not have issues. If you rewrote the history, you will have to force push

(`git push -f`).

→ `git pull [<remote-alias>] [<branch>]`

Fetch commits from the remote branch and apply them on your current branch (conflicts may occur).

# Did you say merge?

```
  A---B---C topic
 /
D---E---F---G master
```

Considering `master` is the current branch:

→ `git merge topic`

```
      A---B---C topic
     /         \
D---E---F---G---H master
```

Merging a branch into another will create a merge commit on the destination branch that will contains all the changes made on the source branch.

# Gotta go fast

If the source branch is an ancestor of the destination branch, there is no need to create a merge commit and git will simply apply the new commits to the destination branch. This is called **fast forward**.

```
      D---E---F topic
      /
A---B---C master
```

→ **git merge topic**

```
A---B---C---D---E---F master, topic
```

If you want to create to merge commit anyway, use **git merge --no-ff topic**

# Merge conflicts

If a file was modified on both branches, git will try its best to manage conflicts, but you will sometimes have to handle them manually.

Here are lines that are either unchanged from the common ancestor, or cleanly resolved because only one side changed.

```
<<<<<< yours:sample.txt
```

```
Conflict resolution is hard;
```

```
let's go shopping.
```

```
=====
```

```
Git makes conflict resolution easy.
```

```
>>>>>> theirs:sample.txt
```

```
And here is another line that is cleanly resolved or unmodified.
```

In the case you have conflicts to fix, stage your modifications (`git add`), and then `git commit` or `git merge --continue`

You can gather the changes of the source branch and discard changes of the destination branch using `git checkout --ours [<file>]`.

You can execute the other way using `git checkout --theirs [<file>]`

# How about rebase?

Rebasing a branch will apply the commits of the source branch on the destination branch. No merge commit will be created.

```
      A---B---C topic
      /
D---E---F---G master
```

Considering `topic` is the current branch:

→ `git rebase master`

```
      A'--B'--C' topic
      /
D---E---F---G master
```

**Be cautious, rebasing rewrites history!**

Notice that `A'`, `B'` and `C'`, even if they contain the same changes as `A`, `B` and `C`, are actually new commits, because you rewrote the history of `topic`.

# Managing conflicts when rebasing

As git will apply one-by-one your commits, it may encounter conflicts.

Just manage them as you would do for a merge, then execute `git rebase --continue`.

If you made a mistake or want to cancel the merge, execute `git rebase --abort`.

# Interactive rebase

Interactive rebase is a powerful tool

A---B---C---D---E---F master

→ `git rebase -i B`

`pick a1b2c3 I am commit C.`

`pick d1e2a3 I am commit D.`

`pick b1c2d3 I am commit E.`

`pick e1a2b3 I am commit F.`

`# Rebase 7212921..f5a245a onto 7212921 (1 command)`

`#`

`# Commands:`

`# p, pick = use commit`

`# r, reword = use commit, but edit the commit message`

`# e, edit = use commit, but stop for amending`

`# s, squash = use commit, but meld into previous commit`

`# f, fixup = like "squash", but discard this commit's log message`

`# x, exec = run command (the rest of the line) using shell`

`# d, drop = remove commit`

# Interactive rebase on a branch

```
    A---B---C topic
    /
D---E---F---G master
```

```
→ git rebase -i master
pick a1b2c3 I am commit A.
pick d1e2a3 I am commit B.
pick b1c2d3 I am commit C.
...
```



# To practice git branching

## Learn Git Branching

# Do you like cherries?

Using `cherry-pick`, you can apply changes of a commit to your branch.

```
A---B---C topic  
/  
D---E---F---G master
```

Considering the active branch is `topic`:

```
→ git cherry-pick F
```

```
A---B---C---F' topic  
/  
D---E---F---G master
```

# Stashing

→ `git stash`

Save both staged & unstaged modifications for later into the stash stack

→ `git stash pop`

Re-apply the last saved modifications

→ `git stash list`

List saved modifications

# Ignoring modifications

Add a `.gitignore` to the root of your repository to ignore some changes

```
# .gitignore
logs/*
.env.local
.idea/*
```

A useful tool: [gitignore.io](https://gitignore.io)

Example: [gitignore.io/api/osx,react,symfony,phpstorm](https://gitignore.io/api/osx,react,symfony,phpstorm)

Useful: [GitHub cheatsheet](#)

# Practical work

Starting from this repository [gitlab.com/lewis\\_mcfly/git-branches-practical-work](https://gitlab.com/lewis_mcfly/git-branches-practical-work)

Only using **merge**, add the B, C and D features to master.

Result should like this:

```
* 6915ddc - (HEAD -> master) Merge branch 'the-b-feature' (2 seconds ago) <Lucas Moisan>
|\
| * 7ca51cb - (the-b-feature) Merge branch 'the-c-feature' into the-b-feature (37 seconds ago) <Lucas Moisan>
| |\
| * f065983 - (origin/the-c-feature, the-c-feature) Add a C feature on top of the B feature. (13 minutes ago) <Lucas Moisan>
| * | 60633bd - (origin/the-b-feature) Fix a bug in the B feature. (12 minutes ago) <Lucas Moisan>
| | /
| * 66ad22b - Add the B feature. (16 minutes ago) <Lucas Moisan>
* | f91852c - Merge branch 'the-d-feature' (19 seconds ago) <Lucas Moisan>
| \ \
| * | 7a78bd0 - (origin/the-d-feature, the-d-feature) Refine the D feature. (4 minutes ago) <Lucas Moisan>
| * | cf2791f - Add the D feature. (10 minutes ago) <Lucas Moisan>
* | | 78aabbc - (origin/master) Fix another bug. (9 minutes ago) <Lucas Moisan>
| / /
* | ee68c76 - Fix a bug. (15 minutes ago) <Lucas Moisan>
* | e697787 - Add another feature. (18 minutes ago) <Lucas Moisan>
| /
* 1273890 - Initial commit. (19 minutes ago) <Lucas Moisan>
```

To display the list of distant branches:

→ `git branch -r`

# Practical work

Starting from this repository [gitlab.com/lewis\\_mcfly/git-branches-practical-work](https://gitlab.com/lewis_mcfly/git-branches-practical-work)

Only using **rebase**, add the B, C and D features to master.

Result should like this:

```
* d21b800 - (HEAD -> master, the-b-feature) Add a C feature on top of the B feature. (9 seconds ago) <Lucas Moisan>
* 60ab576 - Fix a bug in the B feature. (9 seconds ago) <Lucas Moisan>
* 806db74 - Add the B feature. (9 seconds ago) <Lucas Moisan>
* 4c92397 - (the-d-feature) Refine the D feature. (2 minutes ago) <Lucas Moisan>
* 0f84782 - Add the D feature. (2 minutes ago) <Lucas Moisan>
* 78aabbcc - (origin/master) Fix another bug. (15 minutes ago) <Lucas Moisan>
* ee68c76 - Fix a bug. (21 minutes ago) <Lucas Moisan>
* e697787 - Add another feature. (24 minutes ago) <Lucas Moisan>
* 1273890 - Initial commit. (25 minutes ago) <Lucas Moisan>
```

# Finding bugs with bisect

- `git bisect start`
- `git bisect bad <commit>`
- `git bisect good <commit>`
- ...



# Tags

→ `git tag <tag-name>`

Tag the last commit with <tag-name>

Commonly used to tag versions of a project.



**Hosting providers for  
software development  
version control**



**GitLab**



**GitHub**



**Bitbucket**



Centraliser un outil décentralisé...

# Tendance Google : GitHub vs GitLab vs BitBucket

● GitHub ● GitLab ● BitBucket



Dans tous les pays. Cinq dernières années. Recherche sur le Web.

# GitHub x GitLab x BitBucket

## GitHub

*Started in February 2008*

Acquisition by Microsoft June 4, 2018 (US\$7.5 billion)

### **Good to know:**

GitHub Enterprise, GitHub Pages, Gist, Education program, Marketplace, Sponsors...

## GitLab

*Started in 2011*

GitLab moved from Microsoft Azure to Google Cloud Platform in August 11, 2018

### **Good to know:**

GitLab CE: Community Edition and GitLab EE: Enterprise Edition; GitLab Continuous Integration

## BitBucket

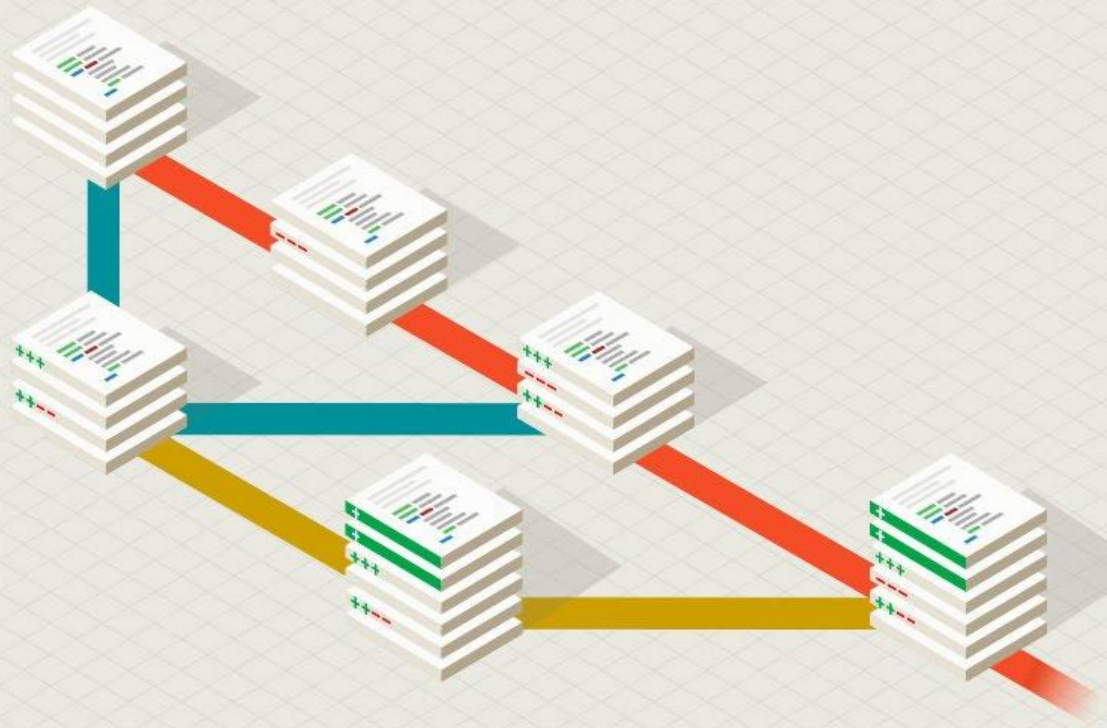
*Started in 2008*

Acquisition by Atlassian in 2010  
Use either Mercurial (since launch till June 1, 2020) or Git (since October 2011)

# Git flow vs Github flow

---

# Git-Flow

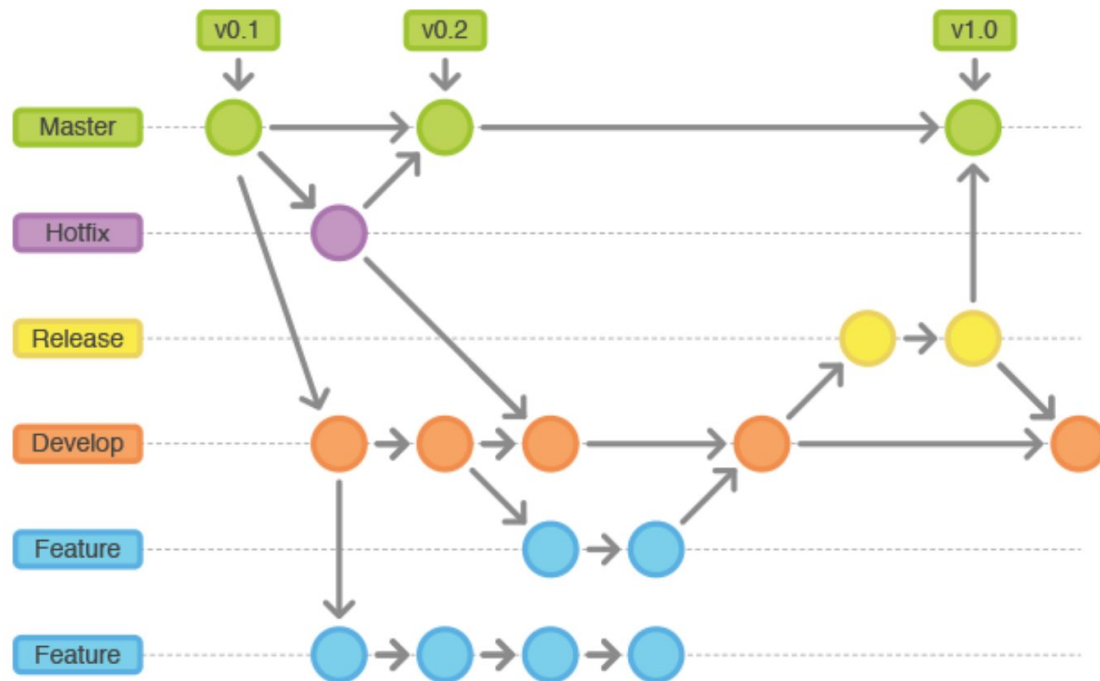


# Git flow : Explication des branches

Branche	Nombre	Origine	TTL	Utilité
master	Unique	NA	Permanente	Code stable, testé et validé potentiellement éligible pour une MEP
hotfix	Aucune/ Plusieurs	master	Correction d'un bug	Corrections des bugs sur le code présent en prod
develop	Unique	master	Permanente	Prochaine version de l'application.
release	Unique	develop	Recette	Corriger les bugs détectés pendant la phase de recette
feature	Plusieurs	develop	Développement d'une feature	Fonctionnalité à embarquer dans la prochaine version de l'application



# Git flow : visualisation de l'arbre



git-flow

# Git flow : Pro / Cons

## Pro

- Développement isolé
  - Regression plus facilement identifiable
  - Descopage possible
  - Expérimentation
- Adapté aux méthodes agile

## Cons

- Complexité
  - 5 branches a surveiller
- Merge multiple
  - Hotfix
  - Release

# TP

En partant du principe qu'un fichier = une feature (eg. ./features/a.txt) et un fichier = un bugfix (eg. ./fixes/fix\_a.txt).

En suivant le git flow, simuler ce changelog:

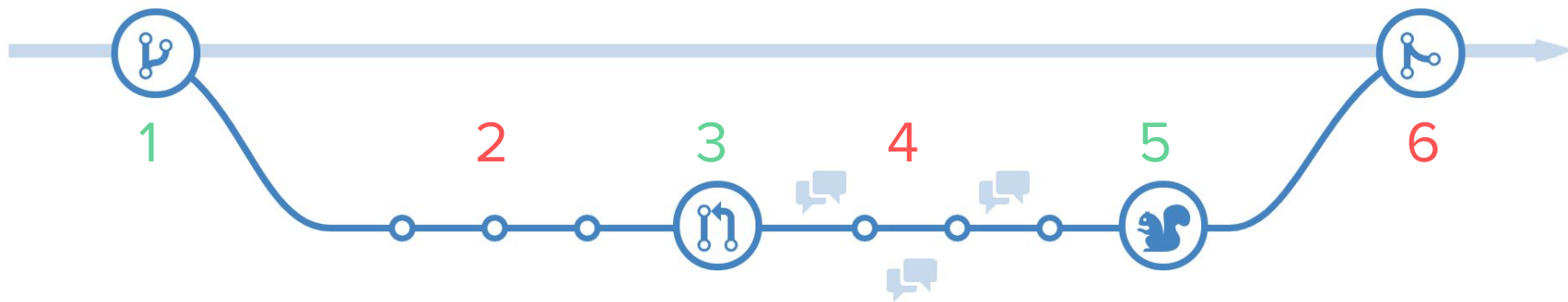
- Création d'une feature A (a.txt)
- Création d'une feature B (b.txt)
- Merge de A
- Fix de B (fix\_b\_1.txt)
- Release de A: pas de modification de fichier
- Merge de B
- Hotfix de A (hotfix\_a\_1.txt)
- Création d'une feature C (c.txt)
- Release de B: ajout d'un fix (fix\_b\_2.txt)
- Merge de C
- Release de C: pas de modification de fichier

# GitHub

## Flow



# GitHub flow : How does it work?



1. Create a branch (with explicit name)
2. Push commits (constantly)
3. Create a *Pull Request*
4. Discuss and review
5. Deploy and test
6. Merge into master

# Allier le Git flow et GitHub flow?

- Pas les mêmes objectifs
- Git flow
  - Maîtriser sa release
  - Déploiement ponctuelle (en fin de release, sauf hotfix)
  - Flow trop complexe pour certains projets
- GitHub flow
  - Dès qu'une feature est prête => merge into master
  - Déploiement régulier (plusieurs fois par jour si besoin)

# SemVer



# SemVer : Semantic Versioning

*In systems with **many dependencies**, releasing new **package versions** can quickly become a nightmare...*

- **Danger of version lock** (the inability to upgrade a package without having to release new versions of every dependent package)

**SemVer** is: Simple set of **rules and requirements** that dictate how **version numbers** are assigned and incremented.

Given a version number **MAJOR.MINOR.PATCH**, increment the:

1. **MAJOR** version when you make incompatible API changes
2. **MINOR** version when you add functionality in a backwards compatible manner
3. **PATCH** version when you make backwards compatible bug fixes.