A large, bold, dark brown word "git" where each letter is a different height and shape, with a small black circle above the "i".

Comprendre le versioning avec Git

14/10/2022



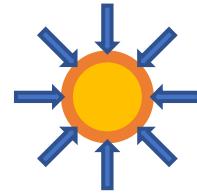
Le versioning

Qu'est ce que le « Version Control »

- Gestion des changements dans des fichiers
- On peut répondre à Qui / Quoi / Quand
- On peut (généralement)
 - Consulter
 - Comparer
 - Restaurer
 - Fusionner

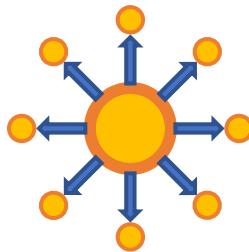
=> Impossible à faire à la main, il faut un VCS (Version Control System)

Versioning : centralized VS distributed



Centralisé

Un seul dépôt de fichiers fait référence

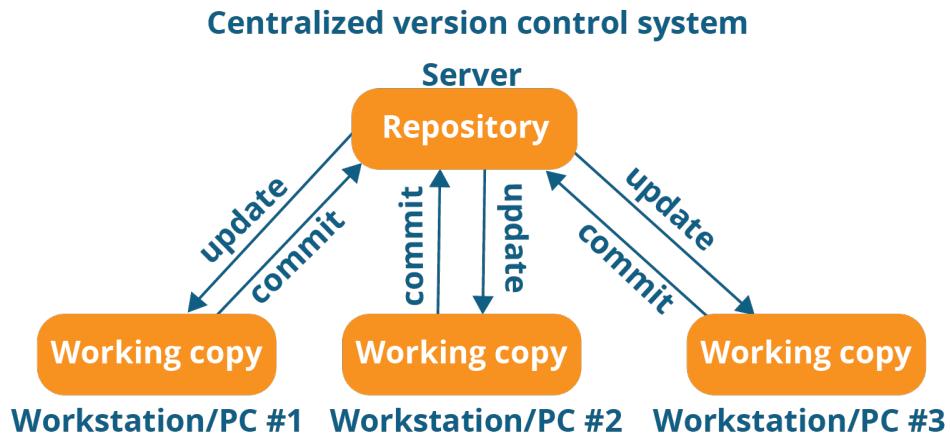


Décentralisé

Chaque client possède une copie des fichiers

Src : medium.com/faun

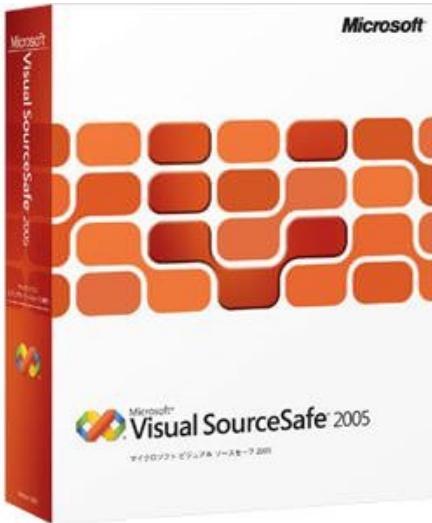
Centralized versioning



- 1 serveur contient toutes les versions de tous les fichiers
- Les clients ont une version des fichiers
- Dès qu'on veut enregistrer une modif, on contacte le serveur
- Concept simple à apprêhender

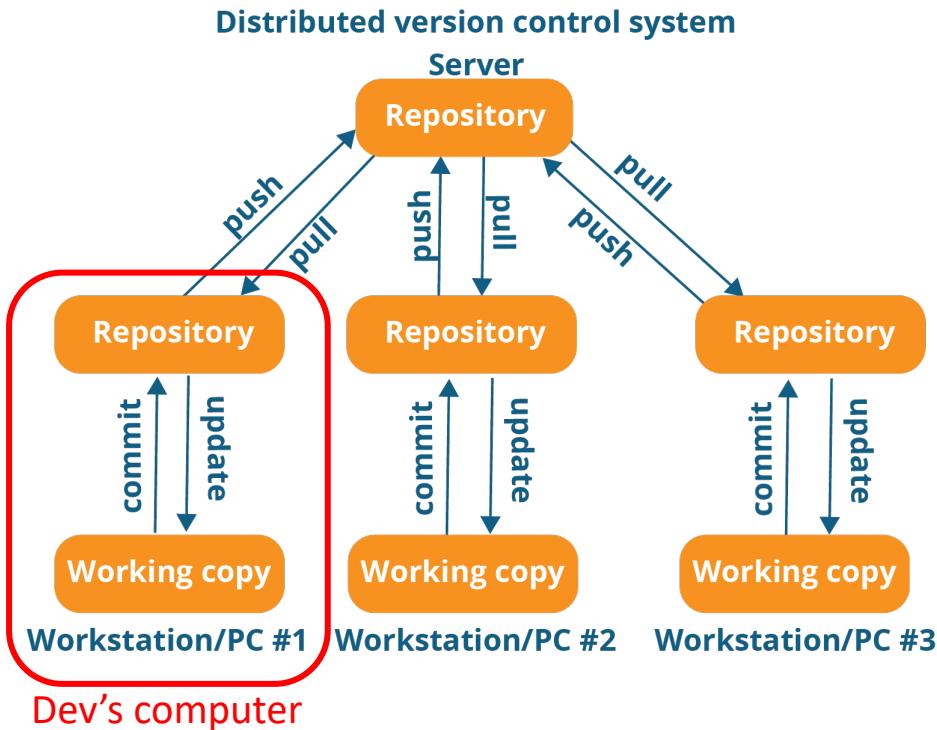
Src : medium.com/faun

Centralized versioning



Perte de popularité de ce modèle de versioning

Distributed versioning



- Serveur et clients ont toutes les versions de tous les fichiers
- Les clients « clone » l'intégralité du repo
- Client peut travailler en local sans réseau

Src : medium.com/faun

Distributed versioning : Pros & Cons

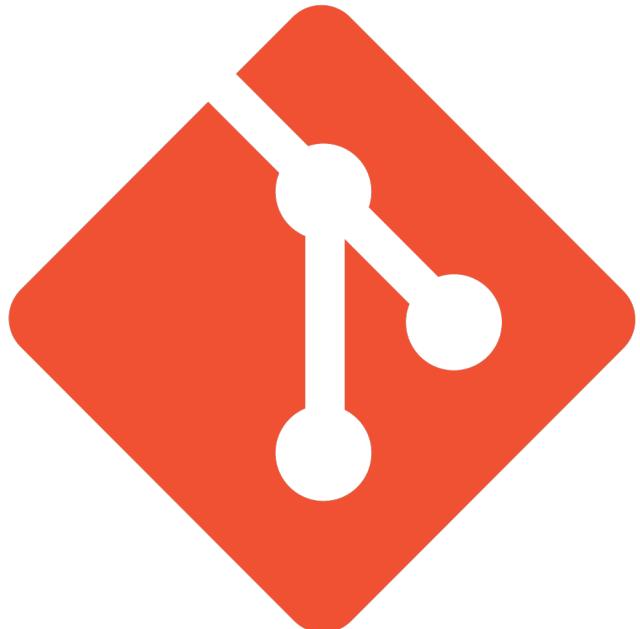


- Travail en local
- Rapidité des actions qui manipulent les fichiers locaux

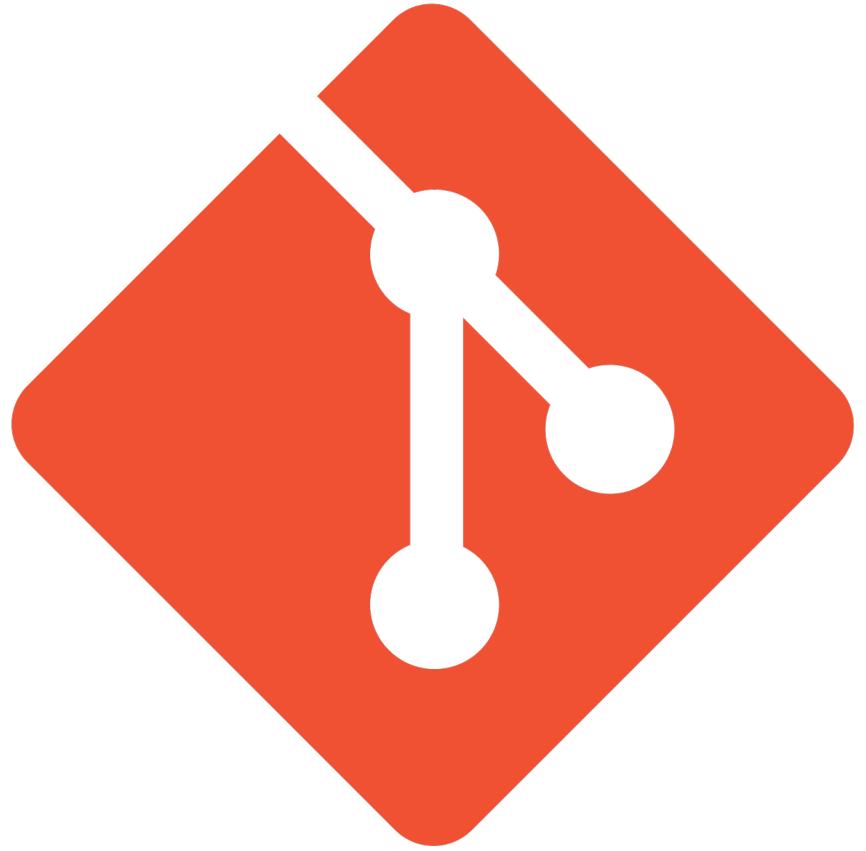


- Pas adapté aux fichiers binaires
- Pas adapté aux très gros historiques

Distributed versioning



mercurial



Git

Benoit Verdier - EPITA

Git : la génèse

Les 90's pour Linu{s,x}

- Internet pas démocratisé
- BBS & Newsgroups & Mail
- Linus applique les patchs Linux
- Linus compile le kernel Linux



Git : la génèse



2002 : Kernel Linux passe sur BitKeeper (décentralisé propriétaire)

2005/04 : Fin de la licence gratuite annoncée pour BitKeeper



2005/04 : Linus Torvalds développe Git



2008 : Lancement de GitHub

2018 : Rachat de GitHub par Microsoft

Les forces de Git

- Adapté au gros projets
- Très bon support du dev non linéaire (avec les branches)
- Grosse communauté
- Ecosystème autour de Git
- Bonnes performances

Git, c'est facile !

Pour un nouveau projet

```
$ git init
```

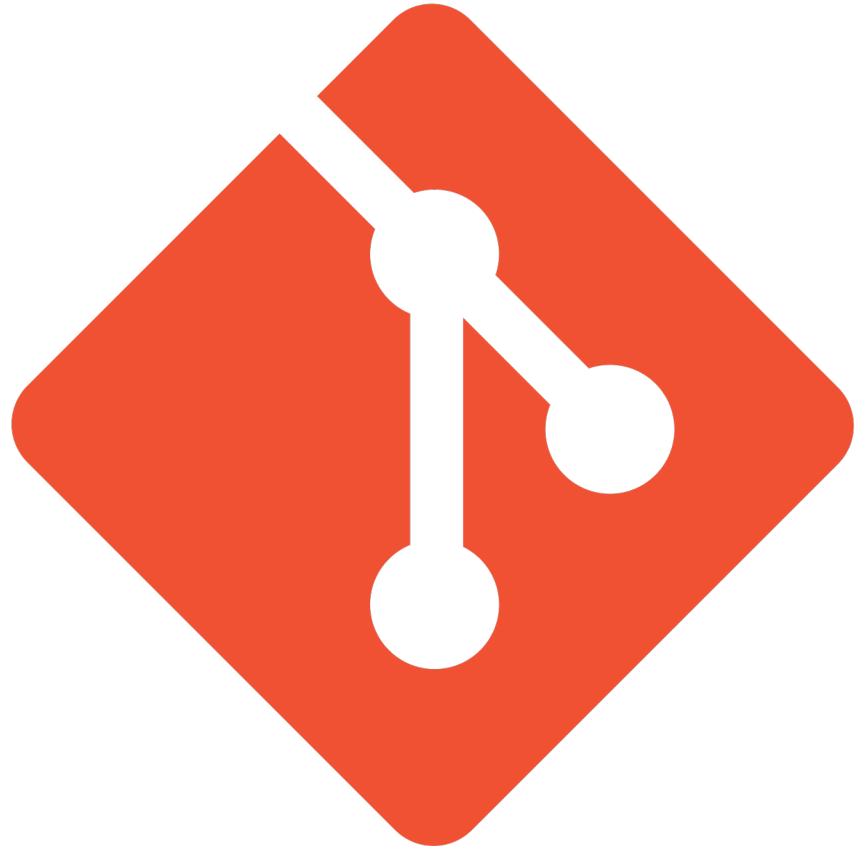
Initialisation d'un projet vide

```
$ git add *.c  
$ git add LICENSE  
$ git commit -m 'initial project version'
```

Création d'un commit avec son message

Pour un projet existant

```
$ git clone https://github.com/libgit2/libgit2
```

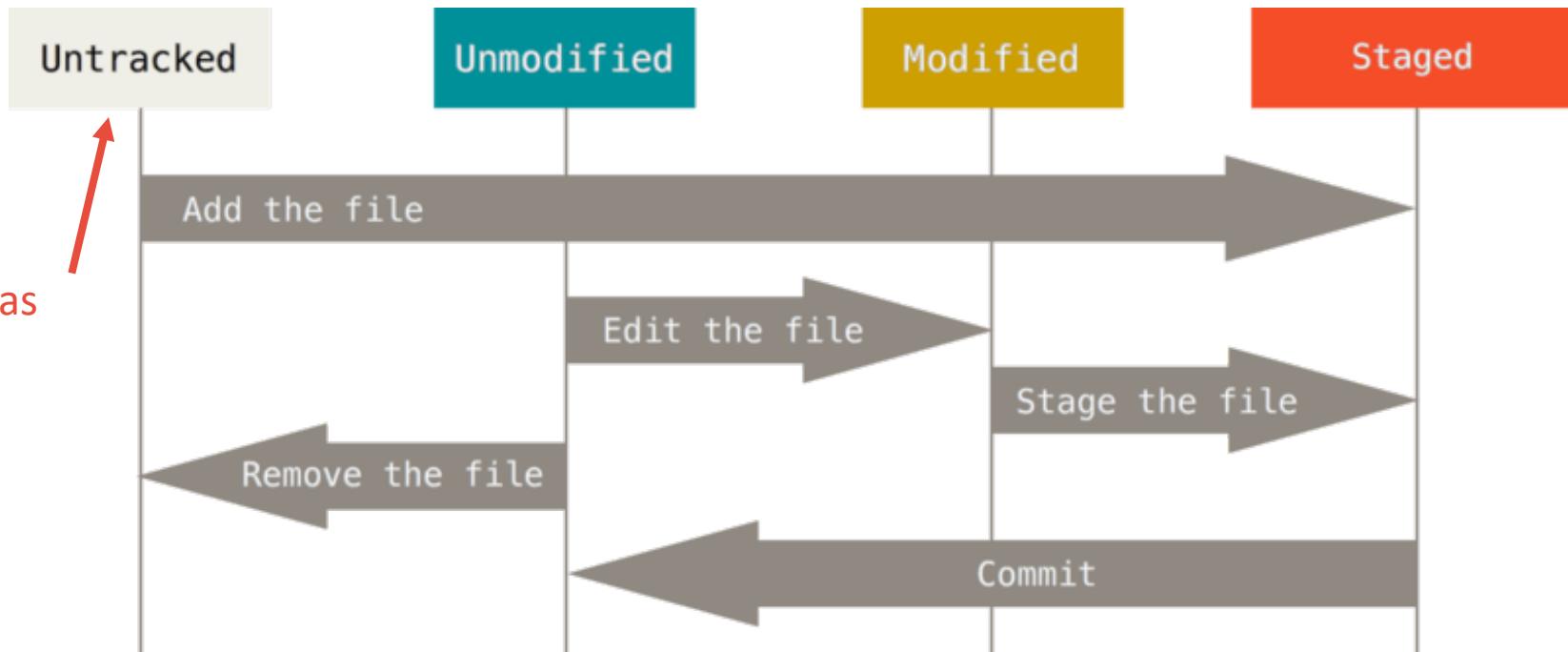


Git : les fondamentaux

Benoit Verdier - EPITA

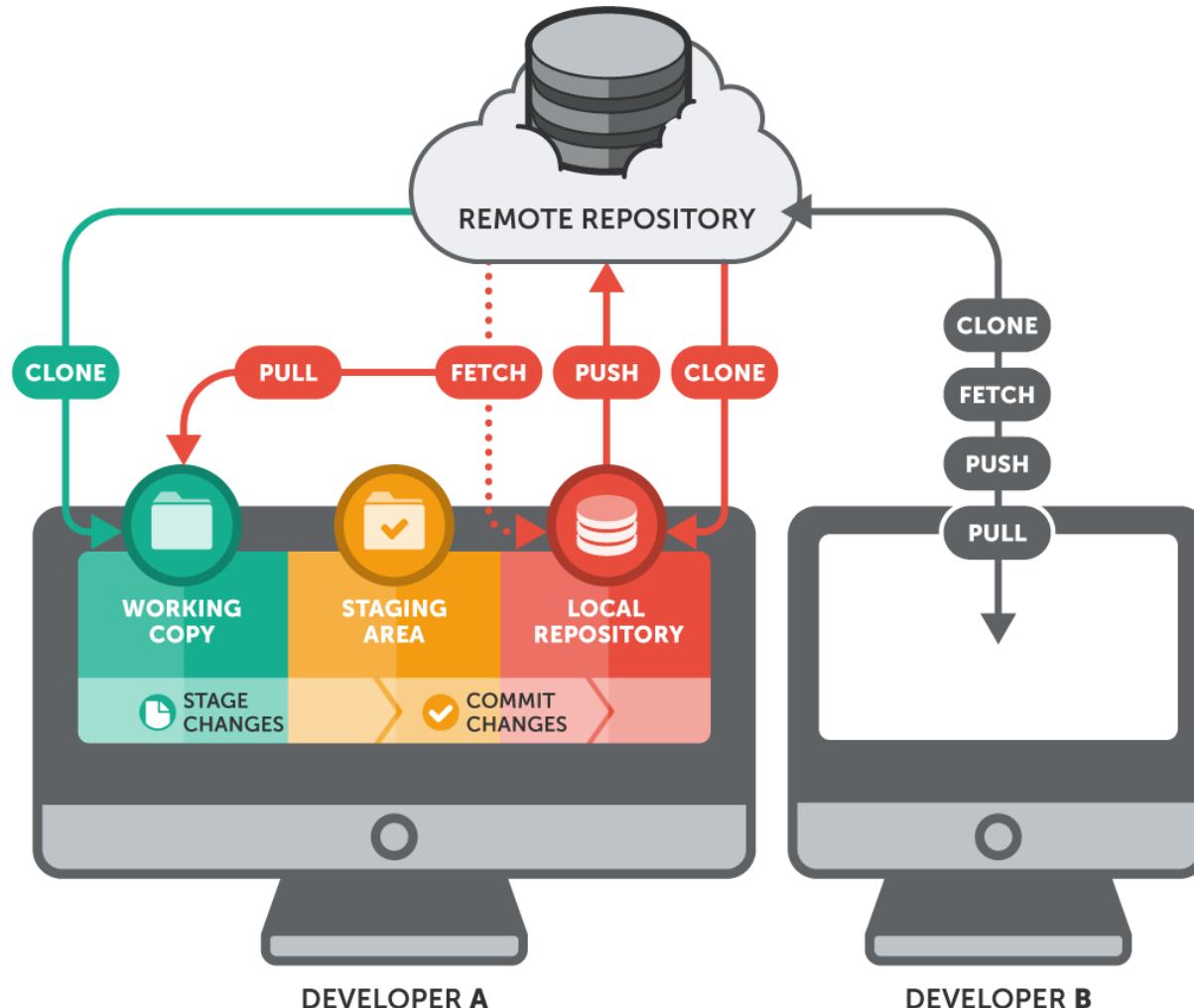
Git : Lifecycle d'un fichier

fichier untracked
= git ne le surveille pas



Src : git-scm.com

Git : local VS remote



Src : git-tower.com

Bonnes pratiques Git

- *Commit* souvent
 - *Commit* à la fonctionnalité ou au correctif
 - *Commit* petit = *revert* facile en cas de problème
 - Transmission + rapide du code aux autres
- Mettre des messages explicites

Bonnes pratiques Git

- Penser à *push/pull* souvent
 - Ca évite les gros conflits
 - *Push* avant de partir -> les autres peuvent continuer bosser
 - *Pull* régulier -> ça évite de rencontrer les bugs déjà corrigés
- Attention, la notion de branche active est commune à toutes les fenêtres/applications/terminaux

Le process

commit => pull => push

Le vrai process : (dev+test+commit) x fois => pull + test => push

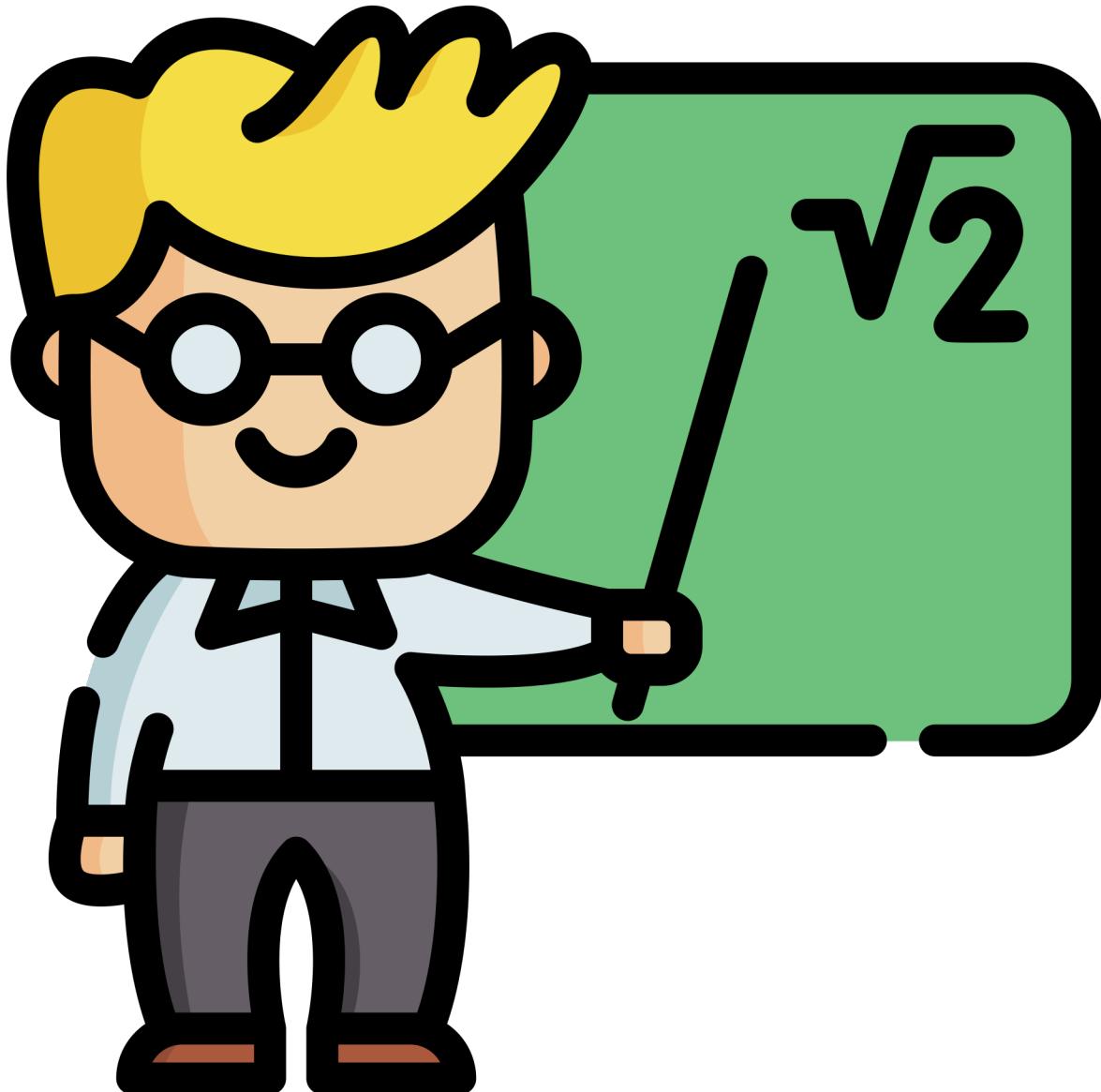
Git : les commandes vitales

- `git init` : crée un nouveau dépôt
- `git clone` : clone un dépôt distant
- `git add` : ajoute des contenus de fichiers au prochain commit (soit des fichiers pas sur le git, soit des fichiers *tracked* et modifiés)
- `git commit` : créé un nouveau commit
- `git push` : publie les nouvelles révisions sur la remote
- `git pull` : récupère les dernières modifications distantes du projet (depuis la Remote)

Git : les commandes vitales

- `git branch` : pour créer et supprimer des branches
- `git checkout` : bascule entre les branches (et les crée si nécessaire).
Permet aussi de restaurer des fichiers à leur état précédent
- `git merge` : fusionner 2 branches (ou plus)

Démo de
GitHub



Mise en pratique

Exo ligne de commande

1. Créer un repo sur GitHub
2. Cloner le repo avec `git clone $URL`
3. Créer en ligne de commande un « `readme.md` » à la racine du repo
4. Vérifier que le nouveau fichier est détecté avec `git status`
5. Ajouter le fichier avec `git add readme.md`
6. Vérifier que le fichier dans la zone de *staging* avec `git status`
7. Créer le commit avec `git commit -m "added a readme"`
8. Vérifier que le commit est créé avec `git status`
9. Envoyer au serveur avec `git push`
10. Confirmer sur GitHub que la modification apparait bien
11. Consulter les logs avec `git log`

GitHub : Parenthèse technique pour le 2FA

2 possibilités pour la ligne de commande ou les apps tierces

- Connexion avec clé SSH

<https://docs.github.com/en/authentication/connecting-to-github-with-ssh>

- Connexion user/password avec Personal Access Token en password

<https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>



Git :Les GUI

Pourquoi utiliser un GUI ?

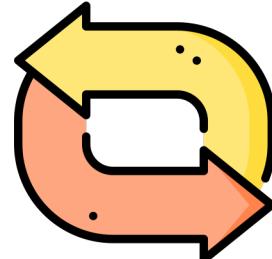


Automatise les procédures (et donc la rigueur)



Facilite

- L'historique
- Les diff
- Les merges
- Les branches
- ...



Indépendance à l'IDE

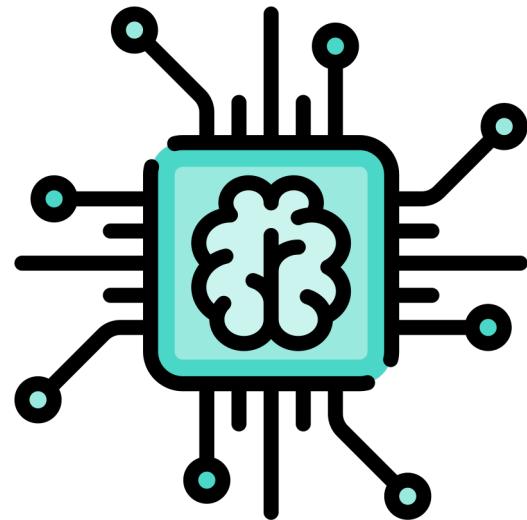
Pourquoi utiliser le CLI ?

- Donne accès à des fonctionnalités avancées
- Fonctionne partout, même sans UI
- Possibilité de scripting



Pourquoi utiliser l'IDE ?

- Facilité d'accès et d'utilisation
- Fonctionnalité distincte
- Intégration de la sémantique du code



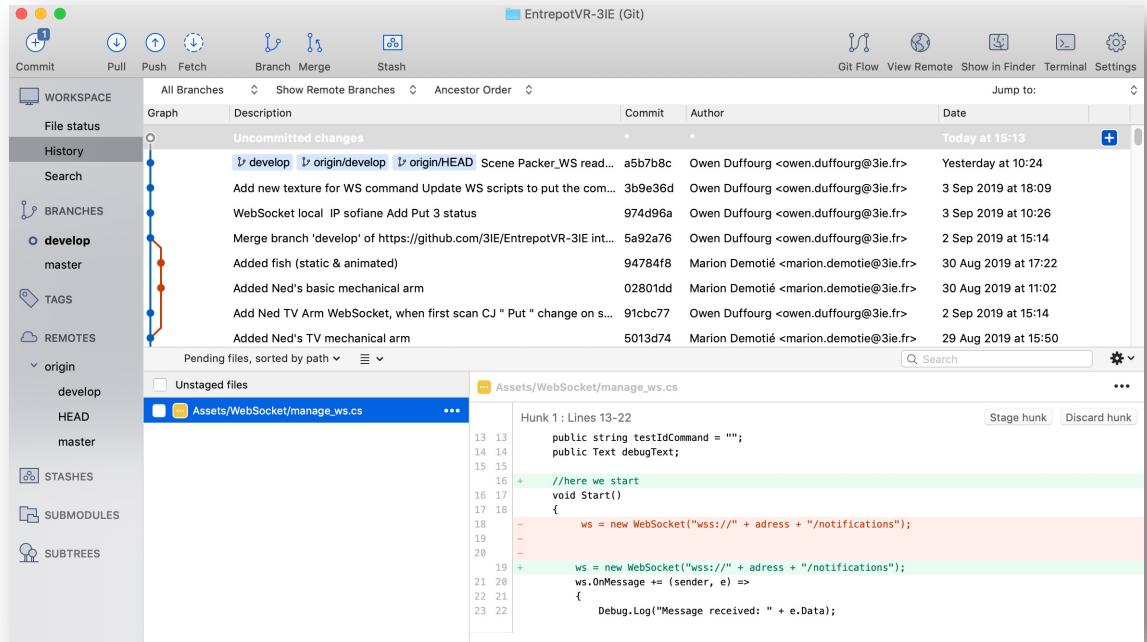


SourceTree

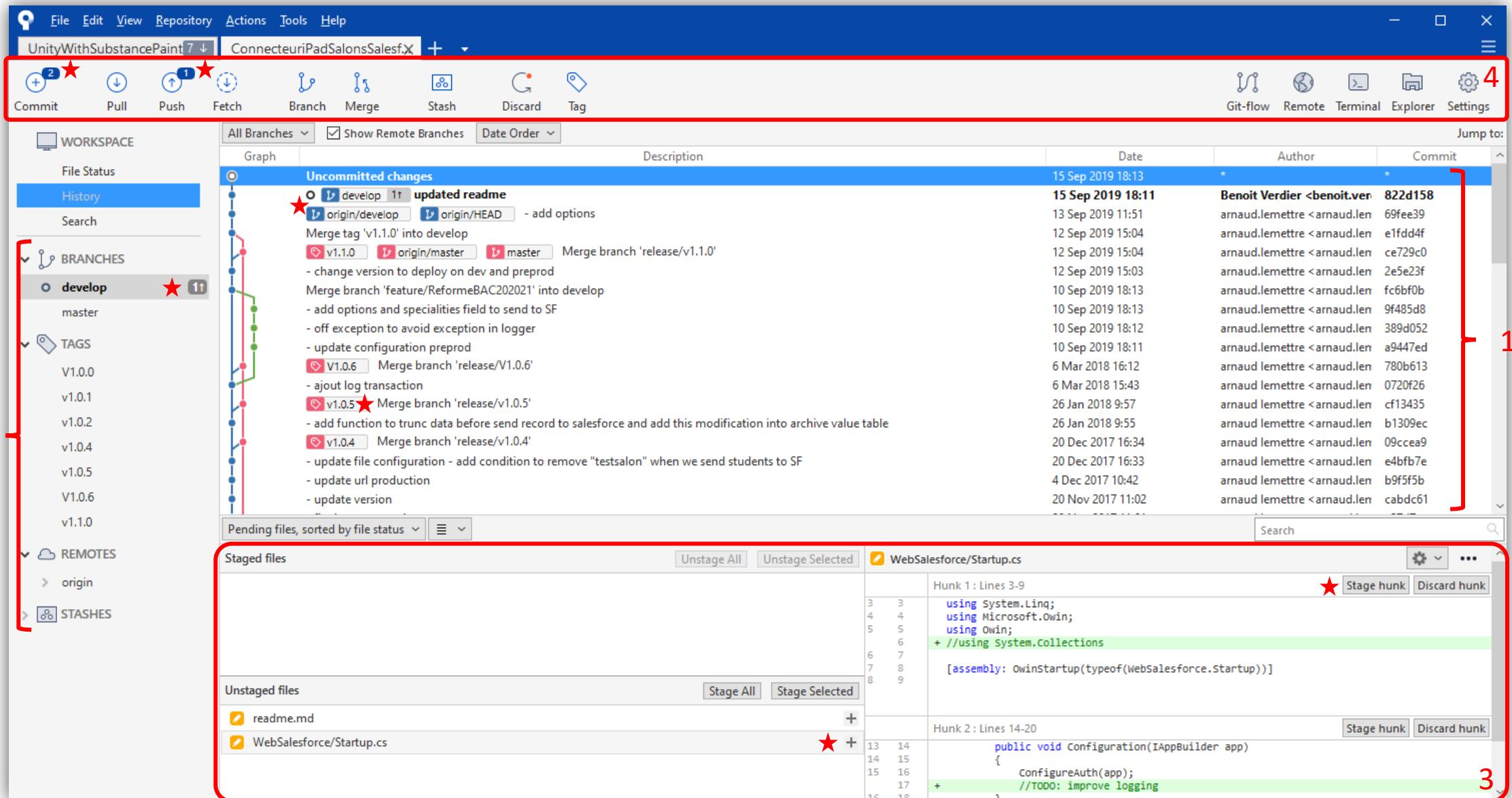
Benoit Verdier - EPITA

SourceTree

- Git GUI pour Windows et Mac
- Gratuit
- Intègre Git
- Connection possible à GitHub, GitLab, ...
- Problème de « credentials » sous Windows

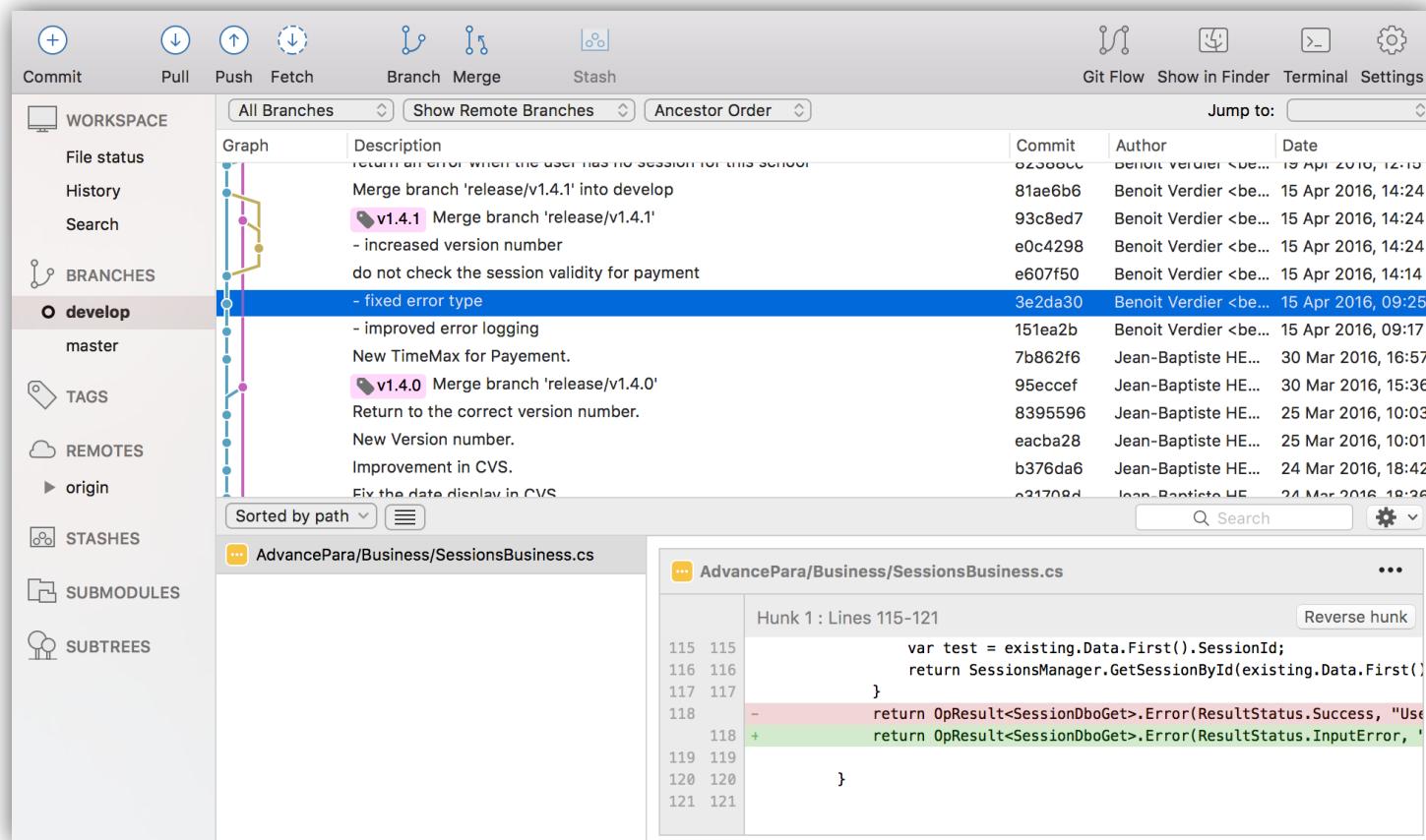


SourceTree : prendre en main l'interface

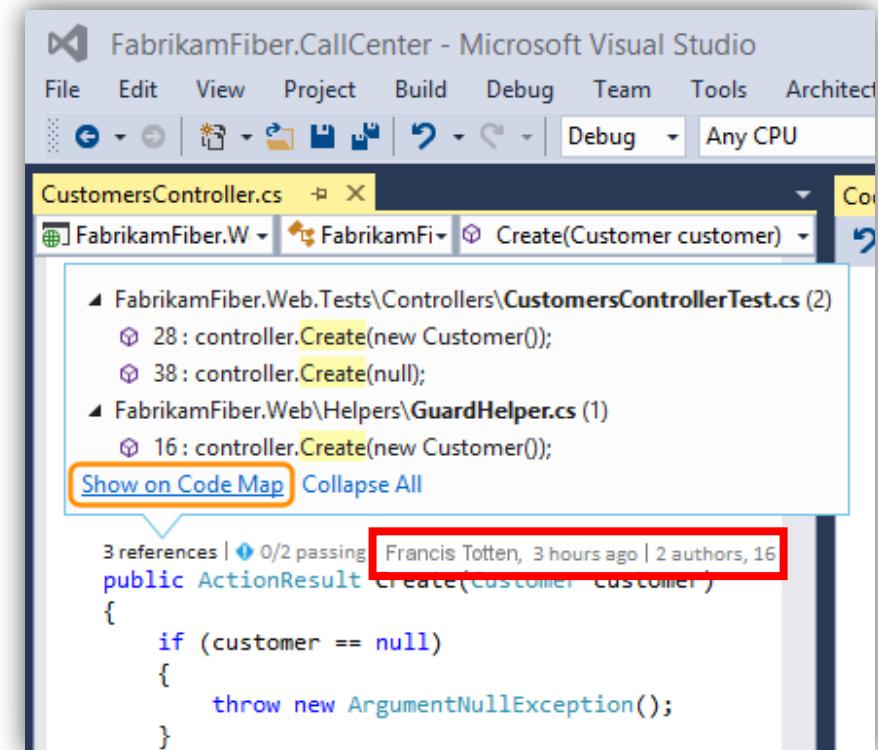


SourceTree : consulter l'historique

Bien dans source tree



mais il y a mieux



Cmdline : Consulter l'historique

Fonctionnel dans le terminal

```
* | 8a2f398 (magicmonkey/db_index) Added an index to aid performance
|/
| * 40a4aed (origin/stubs-are-not-numbers, stubs-are-not-numbers) don't use json_ume
|/
| * 9810699 (richsage/talk-attend, richsage-talk-attend) Update Frisby tests to check
| * 60d3d5c Update "starred" retrieval through Talks API.
| * 5b5860c Patch script changed to "star" from "attend".
| * a5ad013 Update Frisby tests for talk attending/attendee count.
| * 0f8637e Add attending and attendee count into talks-by-speaker output.
| * 87f713d Add attending and attendee count into talk output. This is for the talkBy
| * 06a8bdd Add talk attending_uri to output.
| * 1921d4a Add POST and DELETE handling for talk attendance.
| * a136cac Add talk-attendance value retrieval via GET
| * d541370 Add patch for user_talk_attend table.
|/
* 001df9c Merge branch 'lornajane-event-user-constraint'
|\
| * 8c0b7e3 (origin/event-user-constraint, lornajane/event-user-constraint, lornaja
| |
| * 09725e1 (dbgen-dupes) making the user/event attending checks better
| * 3fd042f add duplicate detection for users attending events
```

Faire un compare

- On a besoin de comparer 2 versions
- Ca s'appelle faire un diff (concept Linux)

Vérifier les modifs en cours

- Diff avec la version précédente
- Toujours le faire avant de commit

Investigation

- Diff entre version X et Y
- Pour comprendre ou « revert »

CmdLine : Faire un compare

```
[15:19] tdd@codewizard:toto (master *) $ git diff
diff --git i/code.rb w/code.rb
index 5a55c17..e32c6de 100644
--- i/code.rb
+++ w/code.rb
@@ -1,8 +1,11 @@
 class DemoController < ApplicationController
- before_filter :find_model, except: [:index, :new, :create]
+ before_filter :find_model, except: [:index, :new, :create]
+
+ private
+
-private
  def find_model
    @model = Demo.find_by_id(params[:id])
+
+  redirect_to({ action: :index }, alert: I18n.t('model.not_found')) unless @model
  end
end
[15:21] tdd@codewizard:toto (master *) $
```

SourceTree : Faire un compare

The screenshot shows the SourceTree application interface. On the left, there's a sidebar with navigation links: WORKSPACE, History, Search, BRANCHES, develop (selected), master, TAGS, REMOTES, origin (selected), develop, HEAD, master, STASHES, SUBMODULES, and SUBTREES. A red arrow points from the text "Modifs non commités" to the "Uncommitted changes" section of the main pane.

The main pane displays a timeline of uncommitted changes. A red box highlights the first change, which is a merge from 'origin/develop' into 'develop'. The commit message is: "Merge branch 'develop' of https://github.com/3IE/EntrepotVR-3IE into develop". The commit hash is 5a92a76, and it was made by Owen Duffourg on 2 Sep 2019 at 15:14. A red box also highlights the file 'Assets/WebSocket/manage_ws.cs' in the pending files list.

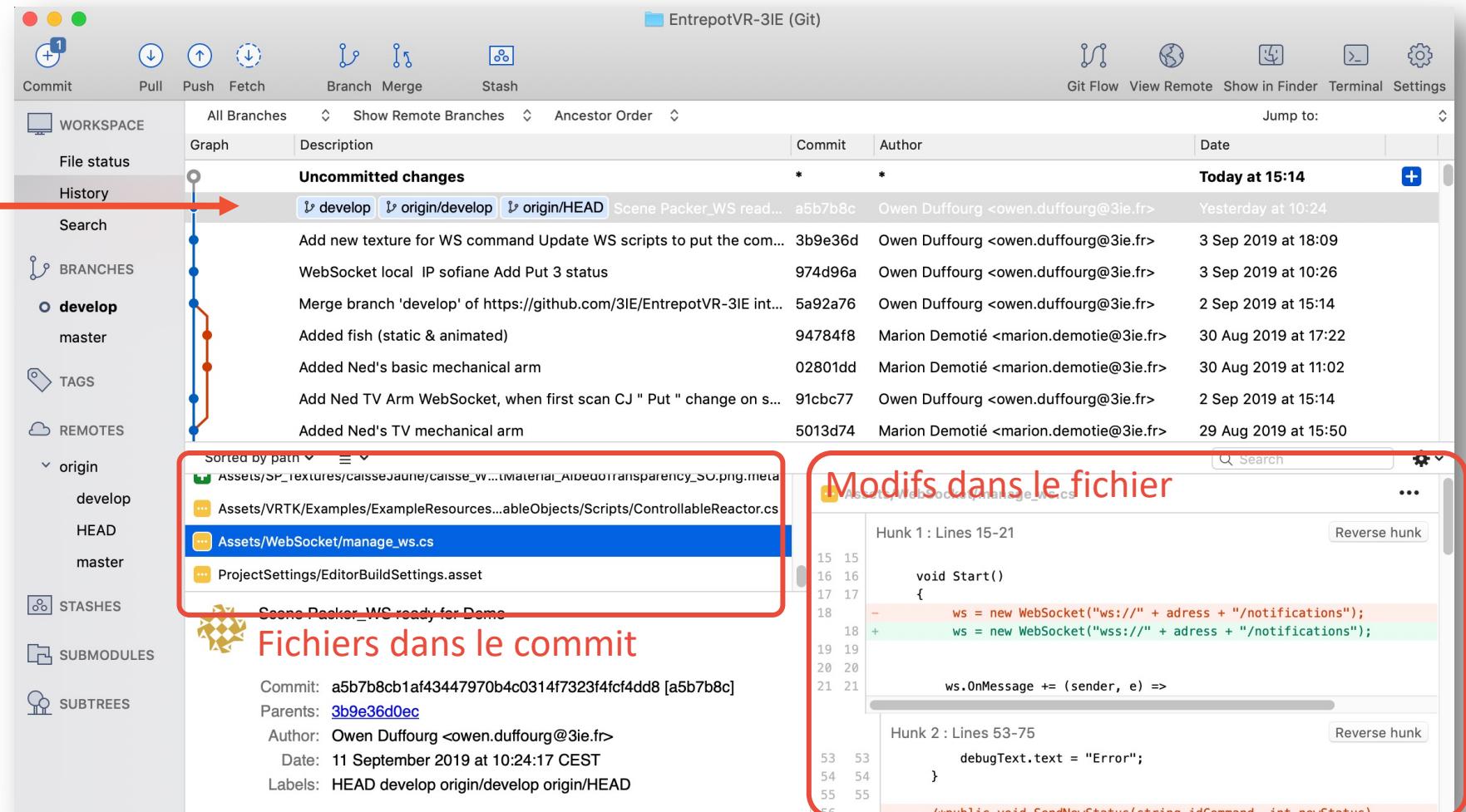
A red callout box with the text "Diff auto sur le fichier sélectionné" is positioned over a diff viewer window. This window shows a code comparison between two versions of the 'manage_ws.cs' file. The left side shows the original code, and the right side shows the modified code with changes highlighted in green and red. The diff viewer includes a search bar, a gear icon, and buttons for "Stage hunk" and "Discard hunk".

Commit	Author	Date
a5b7b8c	Owen Duffourg <owen.duffourg@3ie.fr>	Yesterday at 10:24
3b9e36d	Owen Duffourg <owen.duffourg@3ie.fr>	3 Sep 2019 at 18:09
974d96a	Owen Duffourg <owen.duffourg@3ie.fr>	3 Sep 2019 at 10:26
5a92a76	Owen Duffourg <owen.duffourg@3ie.fr>	2 Sep 2019 at 15:14
94784f8	Marion Demotié <marion.demotie@3ie.fr>	30 Aug 2019 at 17:22
02801dd	Marion Demotié <marion.demotie@3ie.fr>	30 Aug 2019 at 11:02
91cbc77	Owen Duffourg <owen.duffourg@3ie.fr>	2 Sep 2019 at 15:14
5013d74	Marion Demotié <marion.demotie@3ie.fr>	20 Aug 2019 at 15:50

SourceTree : Faire un compare

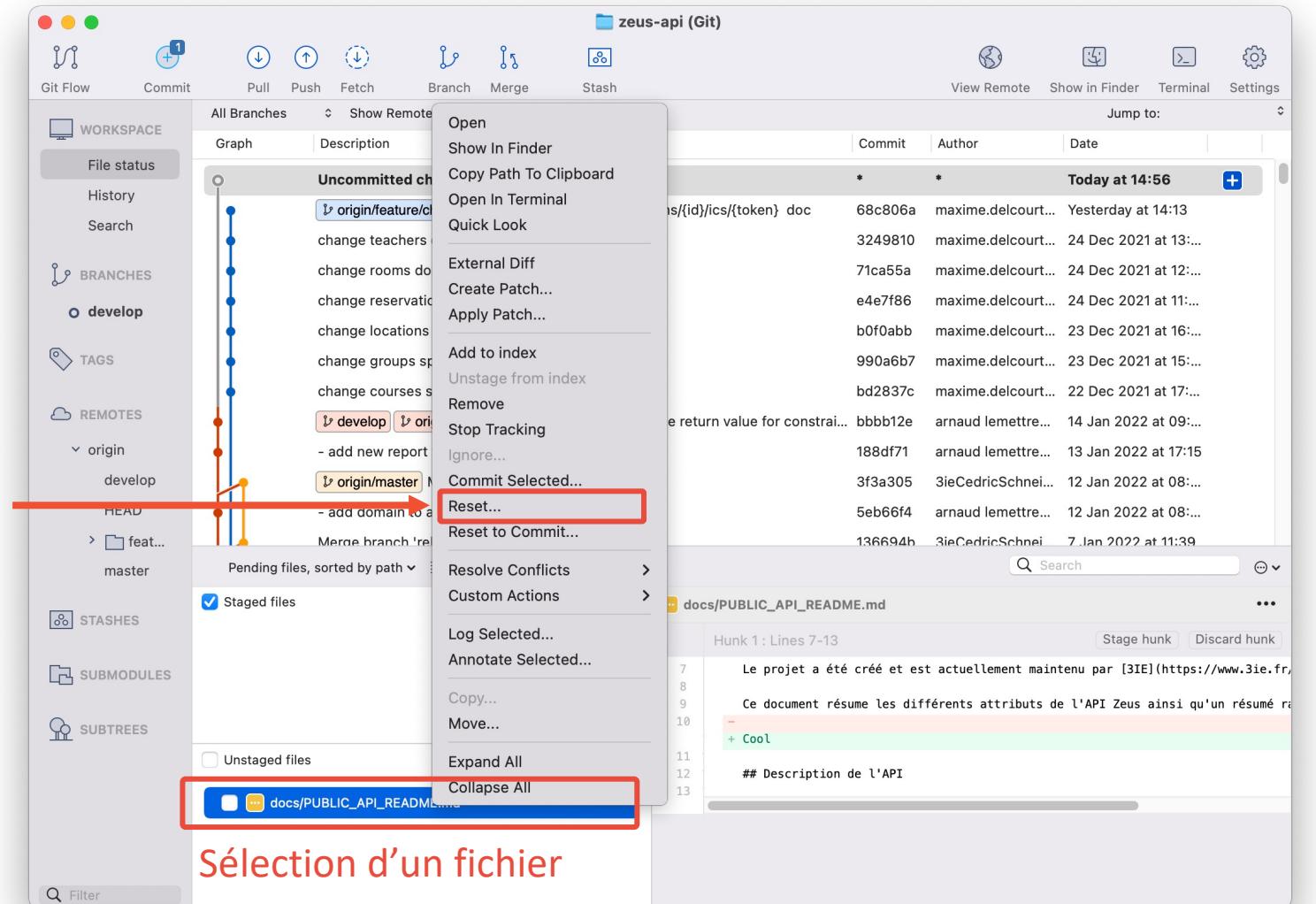
Sélection d'un commit

Historique

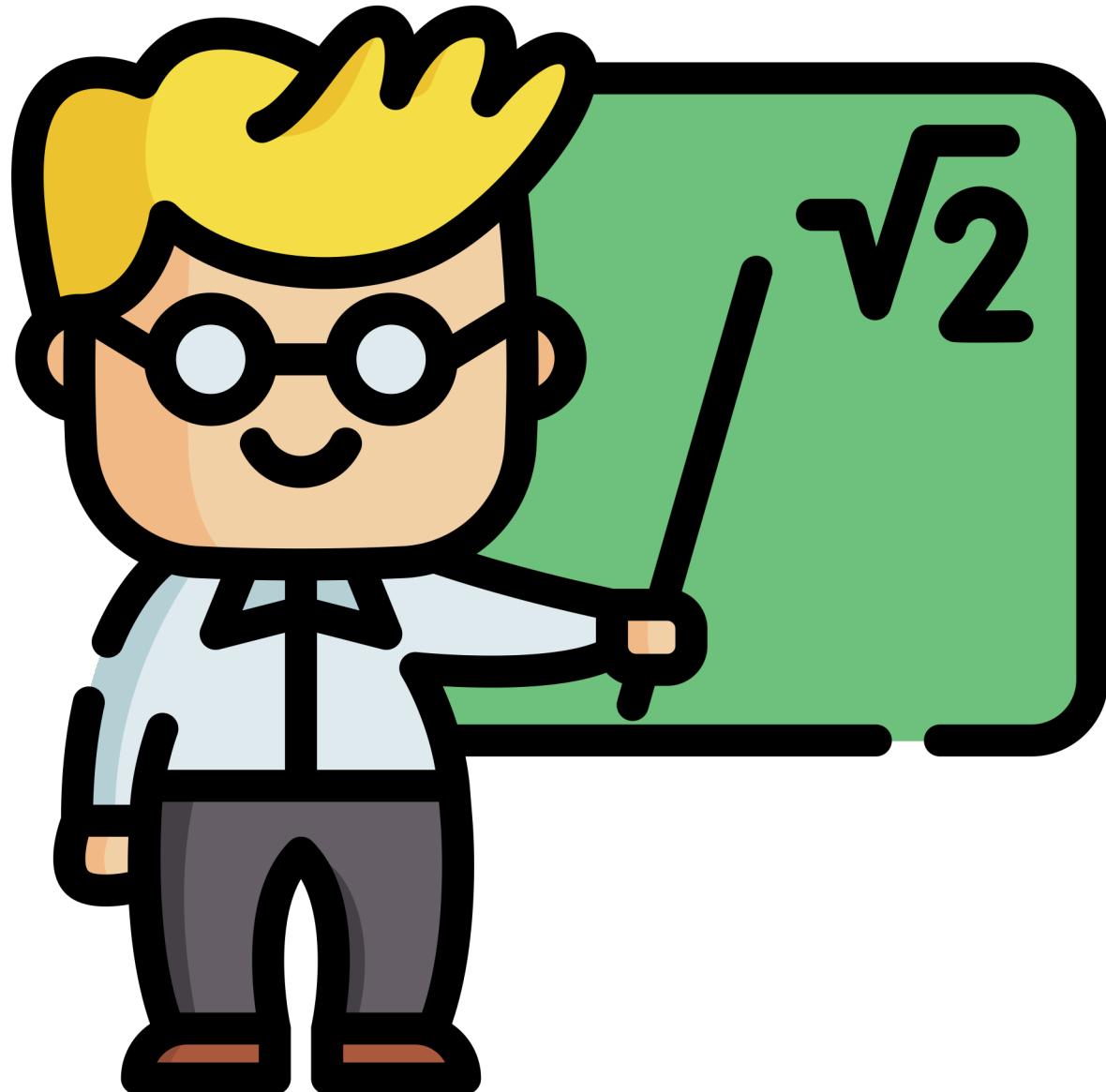


SourceTree : reset une modification

- Permet de supprimer les modifications sur un/des fichier(s)
- Effectue un checkout



Démo
SourceTree



Mise en pratique

SourceTree : Configuration de GitHub

Pour ajouter un repo

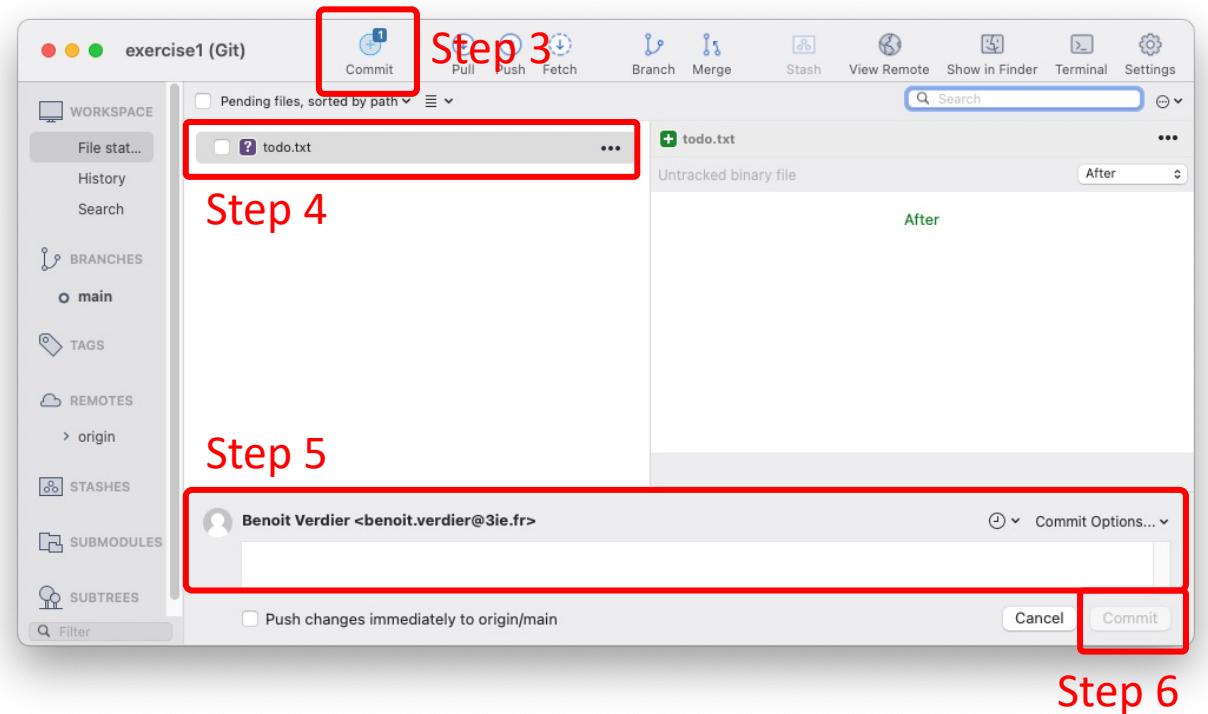
- Soit *clone* de la remote depuis SourceTree
- Soit *drag and drop* du repo (déjà *clone*) dans SourceTree

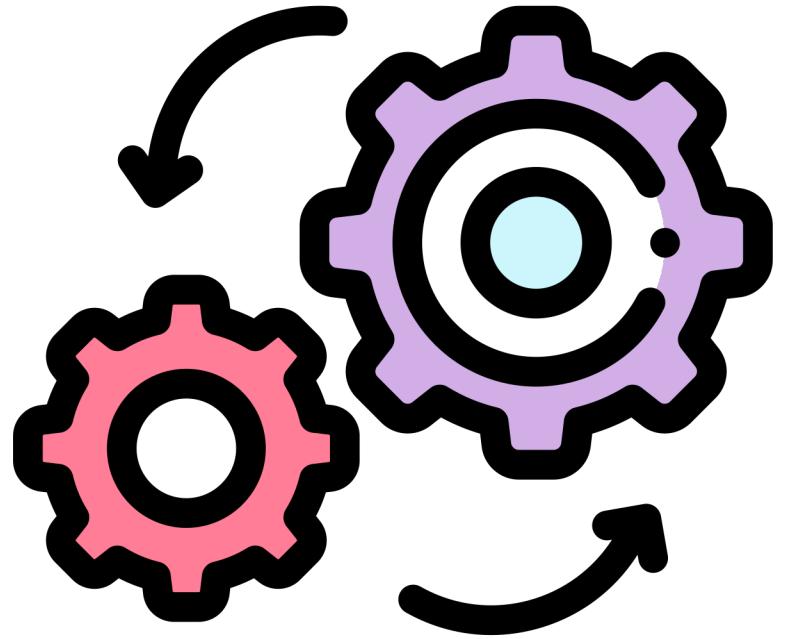
Possibilité de connecter son compte GitHub à SourceTree :

- Allez dans Preferences -> Account

Exercice

1. Reprendre le repo précédemment manipulé en ligne de commande
2. Créer un fichier « todo.txt » à la racine du projet
3. Dans la toolbar de SourceTree, cliquer sur « commit »
4. Cocher le fichier à ajouter
5. Remplir le message de commit
6. Créer le commit
7. Push sur le serveur (toolbar)
8. Vérifier sur GitHub



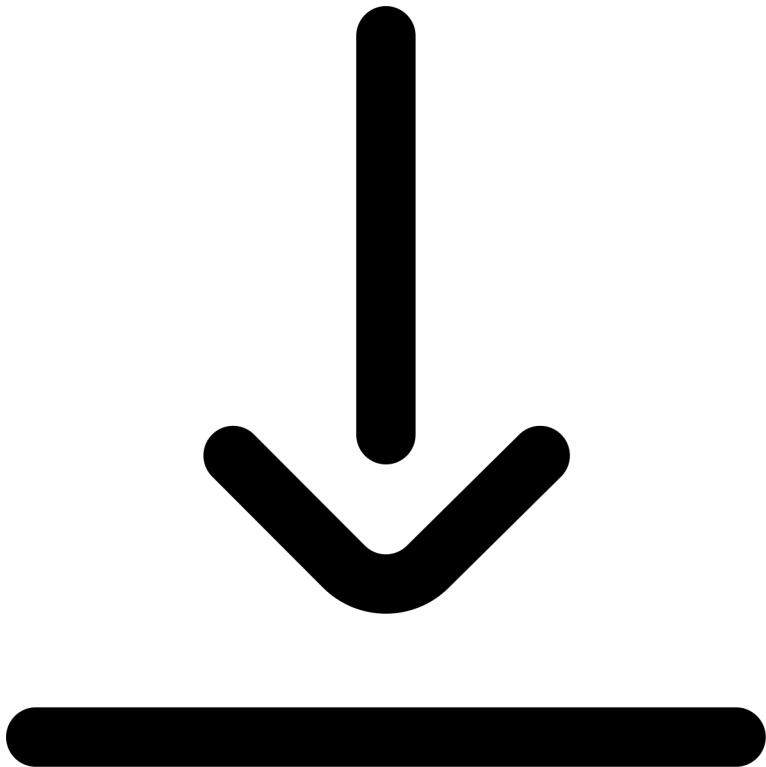


Comment Git stocke-t-il les données ?

Le « .git » folder

=> Dossier caché avec plein de choses

- Configuration du repo git
- Tous les fichiers dans toutes les versions
- Le fichier d'index => staging area
- la HEAD : pointeur vers la branche actuelle (branche active)
- Les refs de commits pour chaque branche
- D'autres configs, logs, ...



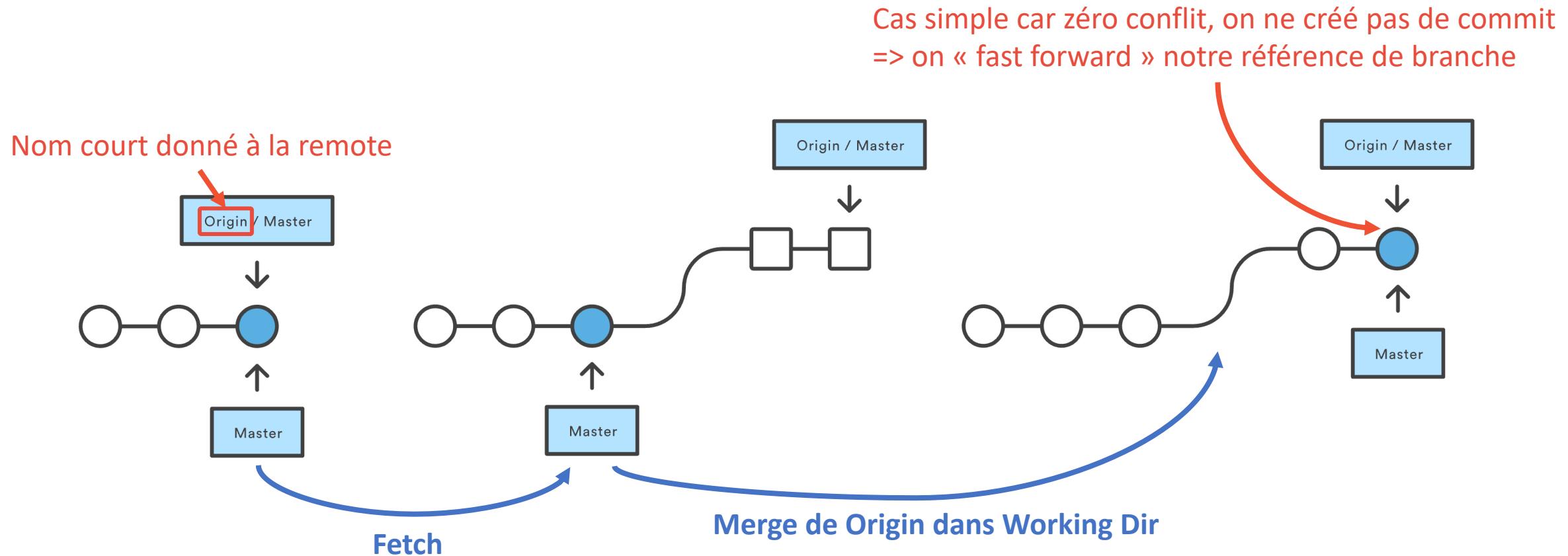
Git : Pull

Git pull

Git pull = git fetch + git merge FETCH_HEAD

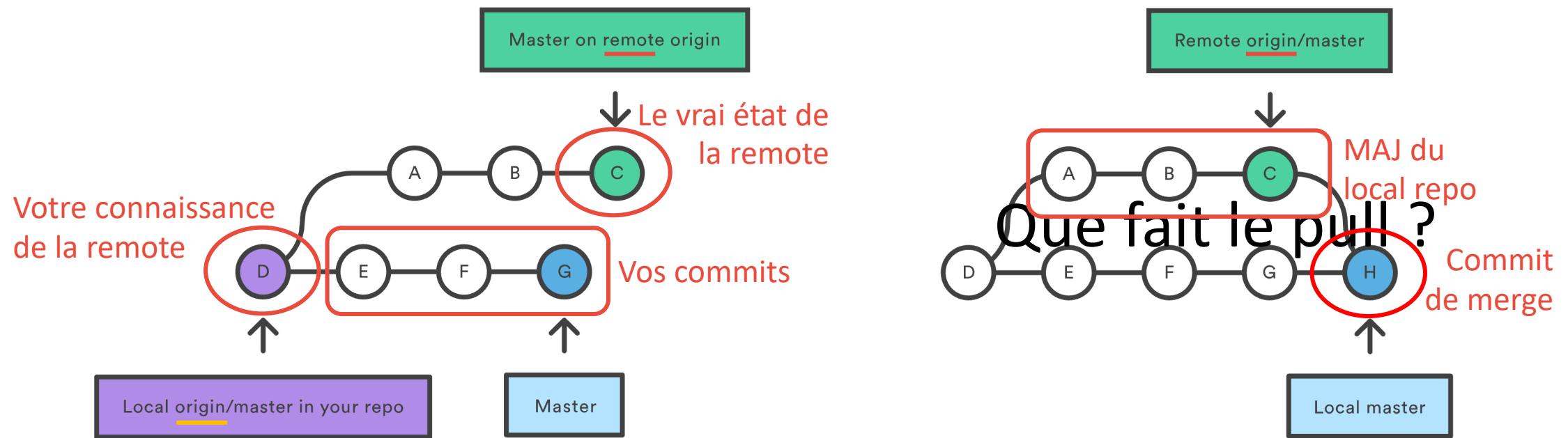
- Fetch : Récupération commits de la remote
- Merge : Merge de la remote dans la branche locale

Processus de pull (cas le plus simple)



Schema by Atlassian is licensed under CC BY 2.5

Processus de pull (cas standard)



Schema by [Atlassian](#) is licensed under CC BY 2.5

Pull : le conflit

- Lors du merge de *origin* dans votre *working dir*
- Conflit entre vous (votre commit) et la remote (commit d'un autre user)
- À résoudre dans le commit de merge
- Le conflit est sur le(s) fichier(s)



→ Même modif => pas de conflit 😊

→ Modif sur des zones différentes => conflits 😞

Pull : le conflit

```
class Main {  
    public static void main(String[] args) {  
        ArrayList<Integer> numbers = new ArrayList<>();  
        numbers.add(2);  
        numbers.add(6);  
        System.out.println("ArrayList1: " + numbers);  
    }  
}
```

Vous

```
class Main {  
    public static void main(String[] args) {  
        ArrayList<Integer> numbers = new ArrayList<>();  
        numbers.add(2);  
        numbers.add(25);  
        System.out.println("ArrayList1: " + numbers);  
    }  
}
```

La remote

Votre modif
Modif sur la remote

```
class Main {  
    public static void main(String[] args) {  
        ArrayList<Integer> numbers = new ArrayList<>();  
        numbers.add(2);  
        <<<<< HEAD  
        numbers.add(6);  
        =====  
        numbers.add(25);  
        >>>>> 88d3d695f945d75c7345fa96222041b001d26596  
        System.out.println("ArrayList1: " + numbers);  
    }  
}
```

Votre fichier suite au merge
=> Il faut résoudre le conflit



Le conflit en GUI

Résoudre un conflit en GUI

```
clock - /Users/Taylor/bin                                         clock vs. clock-backup

CLOCKED_IN=`head -n 1 $TIMECARD | cut -c 10`                  7 → CLOCKED

if [ ${CLOCKED_IN} = "0" ]; then                                8 → if [ ${CLOCKED_IN} = "0" ]; then
    # Record clock status on timecard (IN = 1)                # Record clock status on timecard (IN = 1)
    TIME_IN="date "+%Y/%m/%d %H:%M:%S"                         # TIME_IN="date "+%Y/%m/%d %H:%M:%S"
    echo "In - $TIME_IN" >> $TIMECARD                           echo "In - $TIME_IN" >> $TIMECARD

    # Change to clocked in --> 1                            9 → # Change to clocked in --> 1
    sed -i '' 's>Status: 0/Status: 1/' $TIMECARD               sed -i '' 's>Status: 0/Status: 1/' $TIMECARD

    # Tell the user the clock status (in color)                10 → # Tell the user the clock status (in color)
    echo -e "\033[1;32mStatus - Clocked In\033[0m"           echo -e "\033[1;32mStatus - Clocked In\033[0m"
    echo -e "\033[0;37m$TIME_IN\033[0m"                          echo -e "\033[0;37m$TIME_IN\033[0m"
    echo

else                                                         11 → else
    # Grab timestamp for in-clock                            12 → # Grab timestamp for in-clock
    TIME_IN=`tail -n 1 $TIMECARD`                             # TIME_IN=`tail -n 1 $TIMECARD`
    TIME_OUT="Out - `date "+%Y/%m/%d %H:%M:%S"``             TIME_OUT="Out - `date "+%Y/%m/%d %H:%M:%S``

    # Record clock status on timecard (OUT = 0)              13 → # Record clock status on timecard (OUT = 0)
    echo $TIME_OUT >> $TIMECARD                            echo $TIME_OUT >> $TIMECARD

    # Calculate time difference for total hours clocked in   14 → # Calculate time difference for total hours clocked in
    HOURS=$((Session Length - `timediff.rb "$TIME_IN" "$TIME_OUT"``))  # HOURS=$((Session Length - `timediff.rb "$TIME_IN" "$TIME_OUT"``))
    echo "$HOURS" >> $TIMECARD                            echo "$HOURS" >> $TIMECARD
    echo >> $TIMECARD                                     echo >> $TIMECARD

    # Change to clocked out --> 0                         15 → # Change to clocked out --> 0
    sed -i '' 's/Status: 1/Status: 0/' $TIMECARD            sed -i '' 's/Status: 1/Status: 0/' $TIMECARD

    # Tell the user the clock status                         16 → # Tell the user the clock status
    echo "Status: Clocked In"                                echo "Status: Clocked In"
    echo

else                                                         17 → else
    # Grab timestamp for in-clock                            18 → # Grab timestamp for in-clock
    TIME_IN=`tail -n 1 $TIMECARD`                             # TIME_IN=`tail -n 1 $TIMECARD`
    TIME_OUT="Out - `date "+%Y/%m/%d %H:%M:%S"``             TIME_OUT="Out - `date "+%Y/%m/%d %H:%M:%S"``

status: 18 differences
```

Mac Default

The screenshot shows the Kaleidoscope IDE interface with two panes. The left pane displays the local JavaScript file 'getting-started.js.LOCAL.17480.js' with several lines highlighted in yellow and purple. The right pane shows the 'Resolved Document' with the corresponding code. A yellow box highlights the section of code that handles the newsletter email address field, and a purple box highlights the section that updates the controller when the email address changes.

```
ches)
    return true;
}
return false;
};

$(document).ready(function() {
  // Give focus to the newsletter text field
  $("#newsletter-email-address").focus(); // Line 334

  // Convert buttons to jQuery UI Buttons so that we get proper active state mouse-tracking
  $("button").button();

  // Mark all headers as collapsible
  $("#scopes li").click(switchScope); // Line 340

  // Update the controller when the email address changes
  $("#newsletter-email-address").bind('keyup change', function() { // Line 343
    var emailAddress = this.value;
    if (gettingStartedController != null && gettingStartedController.setEmailAddress_ != null) {
      gettingStartedController.setEmailAddress_(this.value);
    }
  });
}

root.matchMedia(mediaQuery).matches
return true;
}
return false;
};

$(document).ready(function() {
  // Convert buttons to jQuery UI Buttons so that we get proper active state mouse-tracking
  $("button").button();

  // Update the controller when the email address changes
  $("#newsletter-email-address").bind('keyup change', function() {
    var emailAddress = this.value;
    if (gettingStartedController != null && gettingStartedController.setEmailAddress_ != null) {
      gettingStartedController.setEmailAddress_(this.value);
    }
  });
});
```

Mac Kaleidoscope



WinMerge - [Local - Conflicts - Remote]

File Edit View Merge Tools Plugins Window Help

Local - Conflicts - Remote

Location Pane x Local Conflicts Remote

```
import java.util.Collections;
import java.util.ArrayList;

class Main {
    public static void main(String[] args) {
        // Creating an ArrayList
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        numbers.add(1);
        numbers.add(2);
        numbers.add(3);
        numbers.add(2);
        numbers.add(6);
        System.out.println("ArrayList: " + numbers);

        int count = Collections.frequency(numbers, 2);
        System.out.println("Count of 2: " + count);

        ArrayList<Integer> newNumbers = new ArrayList<Integer>();
        newNumbers.add(5);
        newNumbers.add(6);
        System.out.println("ArrayList2: " + newNumbers);

        boolean value = Collections.disjoint(numbers, newNumbers);
        System.out.println("Two lists are disjoint: " + value);
        System.out.println("End of testing");
    }
}
```

import java.util.Collections;
import java.util.ArrayList;

```
class Main {
    public static void main(String[] args) {
        // Creating an ArrayList
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        numbers.add(1);
        numbers.add(2);
        numbers.add(3);
        numbers.add(2);
        numbers.add(6);
        System.out.println("ArrayList: " + numbers);

        int count = Collections.frequency(numbers, 2);
        System.out.println("Count of 2: " + count);

        ArrayList<Integer> newNumbers = new ArrayList<Integer>();
        newNumbers.add(5);
        newNumbers.add(6);
        System.out.println("ArrayList2: " + newNumbers);

        boolean value = Collections.disjoint(numbers, newNumbers);
        System.out.println("Two lists are disjoint: " + value);
        System.out.println("End of testing");
    }
}
```

import java.util.Collections;
import java.util.ArrayList;

```
class Main {
    public static void main(String[] args) {
        // Creating an ArrayList
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        numbers.add(1);
        numbers.add(2);
        numbers.add(3);
        numbers.add(2);
        numbers.add(25);
        System.out.println("ArrayList: " + numbers);

        int count = Collections.frequency(numbers, 2);
        System.out.println("Count of 2: " + count);

        ArrayList<Integer> newNumbers = new ArrayList<Integer>();
        newNumbers.add(5);
        newNumbers.add(6);
        System.out.println("ArrayList2: " + newNumbers);

        boolean value = Collections.disjoint(numbers, newNumbers);
        System.out.println("Two lists are disjoint: " + value);
        System.out.println("End of testing");
    }
}
```

WinMerge

Utiliser WinMerge avec SourceTree

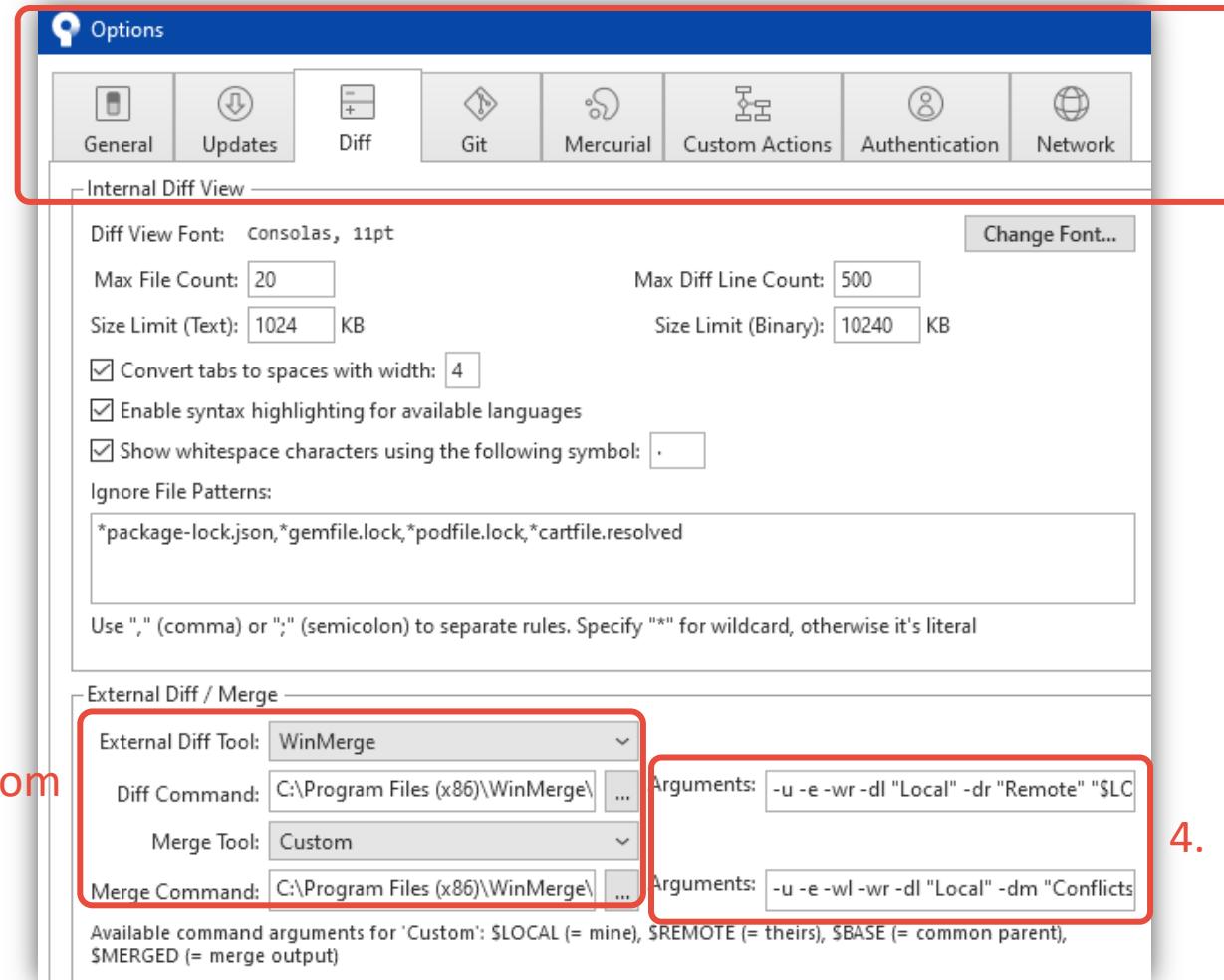


<https://winmerge.org>

Logiciel open source

2. Sélectionner WinMerge/Custom
3. Renseigner le path

1. Options → Diff



4. Arguments à conf

Utiliser WinMerge avec SourceTree

Arguments de diff

```
-u -e -wr -dl "Local" -dr "Remote" "$LOCAL" "$REMOTE"
```

Arguments de merge

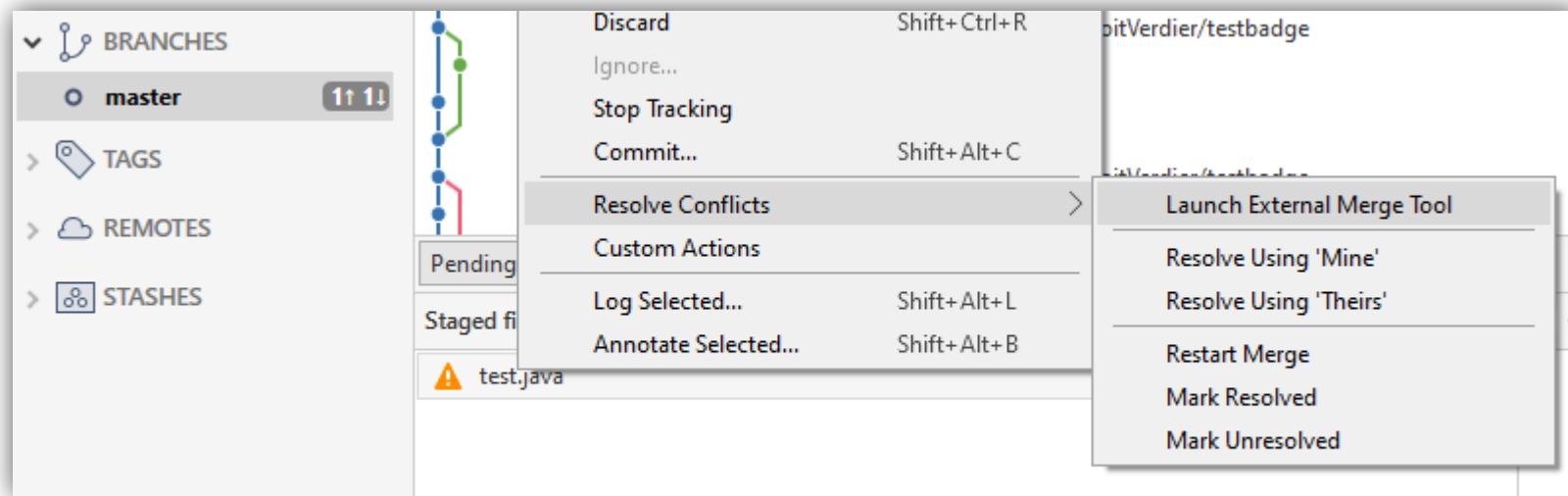
```
-u -e -wl -wr -dl "Local" -dm "Conflicts" -dr "Remote" "$LOCAL" "$BASE" "$REMOTE" -o "$MERGED"
```

Pourquoi ? ⇒ https://manual.winmerge.org/en/Command_line.html

```
WinMergeU /dl leftdesc /dm middledesc /dr rightdesc leftpath middlepath rightpath /o outputpath  
/e : close WinMerge with a single Esc key press  
/u : Prevents WinMerge from adding either path (left or right) to the Most Recently Used list  
/wl : Opens the left side as read-only  
/wr : Opens the right side as read-only
```

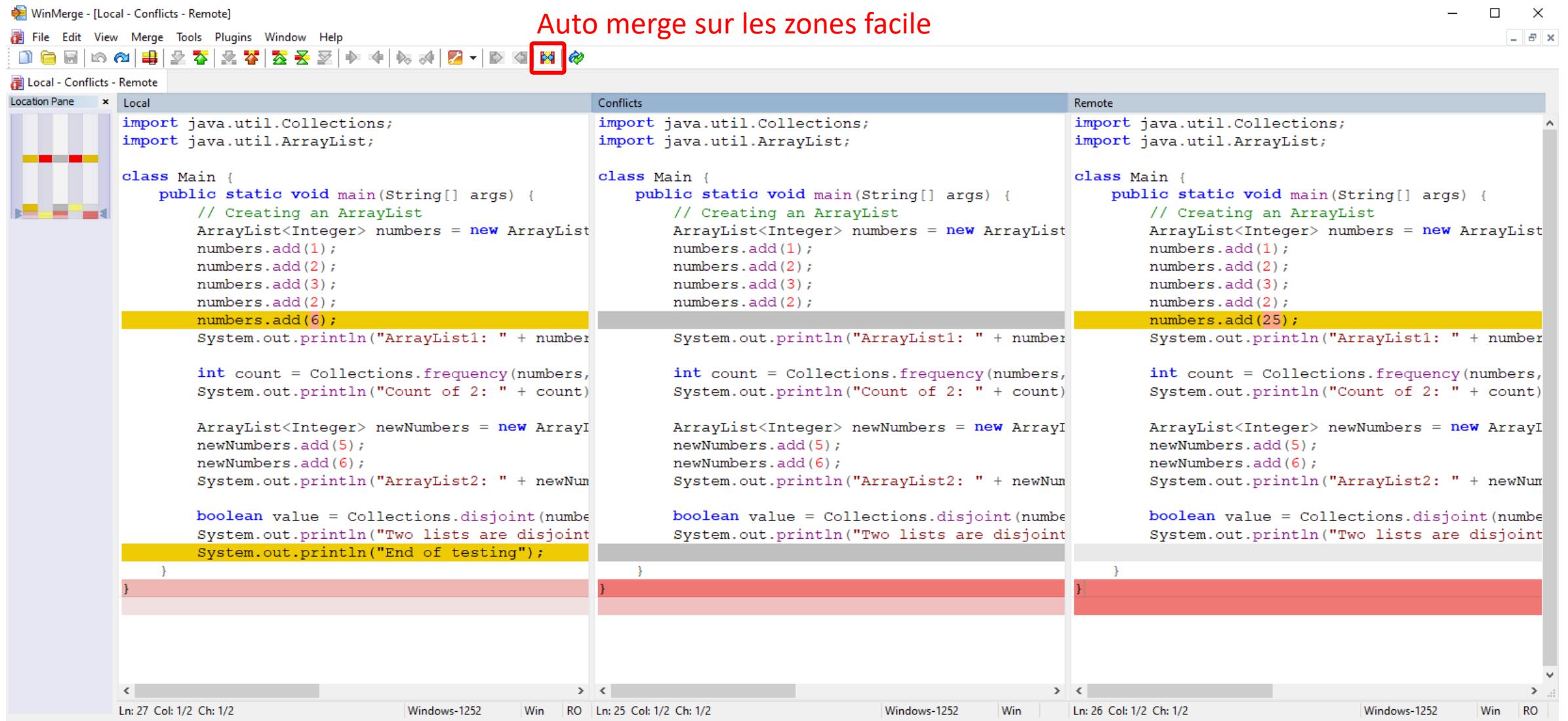
Utiliser WinMerge avec SourceTree

Après le pull, clic droit sur le conflit



Utiliser WinMerge

Auto merge sur les zones facile



The screenshot shows the WinMerge interface comparing three versions of a Java file: Local, Conflicts, and Remote. The Local pane contains the original code. The Conflicts pane shows differences between the Local and Remote versions. The Remote pane contains the updated code from the Conflicts pane. The 'Conflicts' tab is highlighted with a red box.

```
import java.util.Collections;
import java.util.ArrayList;

class Main {
    public static void main(String[] args) {
        // Creating an ArrayList
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        numbers.add(1);
        numbers.add(2);
        numbers.add(3);
        numbers.add(2);
        numbers.add(6);
        System.out.println("ArrayList1: " + numbers);

        int count = Collections.frequency(numbers, 2);
        System.out.println("Count of 2: " + count);

        ArrayList<Integer> newNumbers = new ArrayList<Integer>();
        newNumbers.add(5);
        newNumbers.add(6);
        System.out.println("ArrayList2: " + newNumbers);

        boolean value = Collections.disjoint(numbers, newNumbers);
        System.out.println("Two lists are disjoint: " + value);
        System.out.println("End of testing");
    }
}
```

```
import java.util.Collections;
import java.util.ArrayList;

class Main {
    public static void main(String[] args) {
        // Creating an ArrayList
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        numbers.add(1);
        numbers.add(2);
        numbers.add(3);
        numbers.add(2);
        numbers.add(6);
        System.out.println("ArrayList1: " + numbers);

        int count = Collections.frequency(numbers, 2);
        System.out.println("Count of 2: " + count);

        ArrayList<Integer> newNumbers = new ArrayList<Integer>();
        newNumbers.add(5);
        newNumbers.add(6);
        System.out.println("ArrayList2: " + newNumbers);

        boolean value = Collections.disjoint(numbers, newNumbers);
        System.out.println("Two lists are disjoint: " + value);
        System.out.println("End of testing");
    }
}
```

```
import java.util.Collections;
import java.util.ArrayList;

class Main {
    public static void main(String[] args) {
        // Creating an ArrayList
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        numbers.add(1);
        numbers.add(2);
        numbers.add(3);
        numbers.add(2);
        numbers.add(6);
        System.out.println("ArrayList1: " + numbers);

        int count = Collections.frequency(numbers, 2);
        System.out.println("Count of 2: " + count);

        ArrayList<Integer> newNumbers = new ArrayList<Integer>();
        newNumbers.add(5);
        newNumbers.add(6);
        System.out.println("ArrayList2: " + newNumbers);

        boolean value = Collections.disjoint(numbers, newNumbers);
        System.out.println("Two lists are disjoint: " + value);
        System.out.println("End of testing");
    }
}
```

Utiliser WinMerge

Barre d'actions

Conflit

Réso auto

```
import java.util.Collections;
import java.util.ArrayList;

class Main {
    public static void main(String[] args) {
        // Creating an ArrayList
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        numbers.add(1);
        numbers.add(2);
        numbers.add(3);
        numbers.add(2);
        numbers.add(6);
        System.out.println("ArrayList1: " + numbers);

        int count = Collections.frequency(numbers, 2);
        System.out.println("Count of 2: " + count);

        ArrayList<Integer> newNumbers = new ArrayList<Integer>();
        newNumbers.add(5);
        newNumbers.add(6);
        System.out.println("ArrayList2: " + newNumbers);

        boolean value = Collections.disjoint(numbers, newNumbers);
        System.out.println("Two lists are disjoint: " + value);
        System.out.println("End of testing");
    }
}
```

* Conflicts

```
import java.util.Collections;
import java.util.ArrayList;

class Main {
    public static void main(String[] args) {
        // Creating an ArrayList
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        numbers.add(1);
        numbers.add(2);
        numbers.add(3);
        numbers.add(2);
        numbers.add(6);
        System.out.println("ArrayList1: " + numbers);

        int count = Collections.frequency(numbers, 2);
        System.out.println("Count of 2: " + count);

        ArrayList<Integer> newNumbers = new ArrayList<Integer>();
        newNumbers.add(5);
        newNumbers.add(6);
        System.out.println("ArrayList2: " + newNumbers);

        boolean value = Collections.disjoint(numbers, newNumbers);
        System.out.println("Two lists are disjoint: " + value);
        System.out.println("End of testing");
    }
}
```

Remote

```
import java.util.Collections;
import java.util.ArrayList;

class Main {
    public static void main(String[] args) {
        // Creating an ArrayList
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        numbers.add(1);
        numbers.add(2);
        numbers.add(3);
        numbers.add(2);
        numbers.add(6);
        System.out.println("ArrayList1: " + numbers);

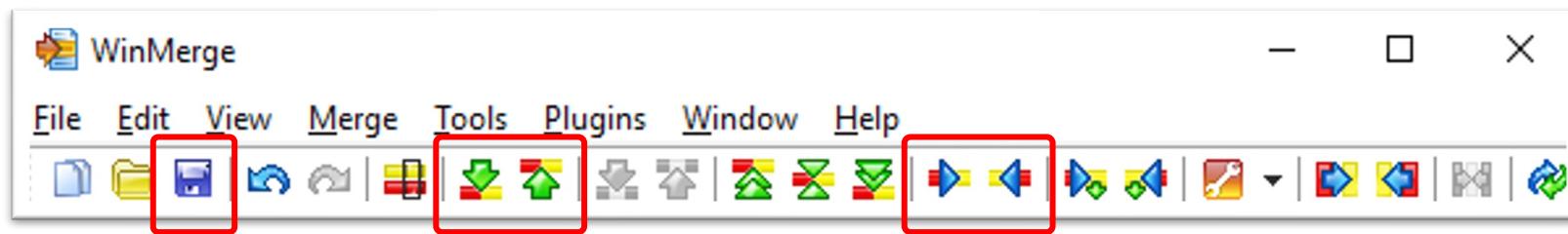
        int count = Collections.frequency(numbers, 2);
        System.out.println("Count of 2: " + count);

        ArrayList<Integer> newNumbers = new ArrayList<Integer>();
        newNumbers.add(5);
        newNumbers.add(6);
        System.out.println("ArrayList2: " + newNumbers);

        boolean value = Collections.disjoint(numbers, newNumbers);
        System.out.println("Two lists are disjoint: " + value);
        System.out.println("End of testing");
    }
}
```

Ln: 12 Col: 24/24 Ch: 24/24 Windows-1252 Win RO Line: 11-12 Windows-1252 Win Ln: 12 Col: 25/25 Ch: 25/25 Windows-1252 Win RO

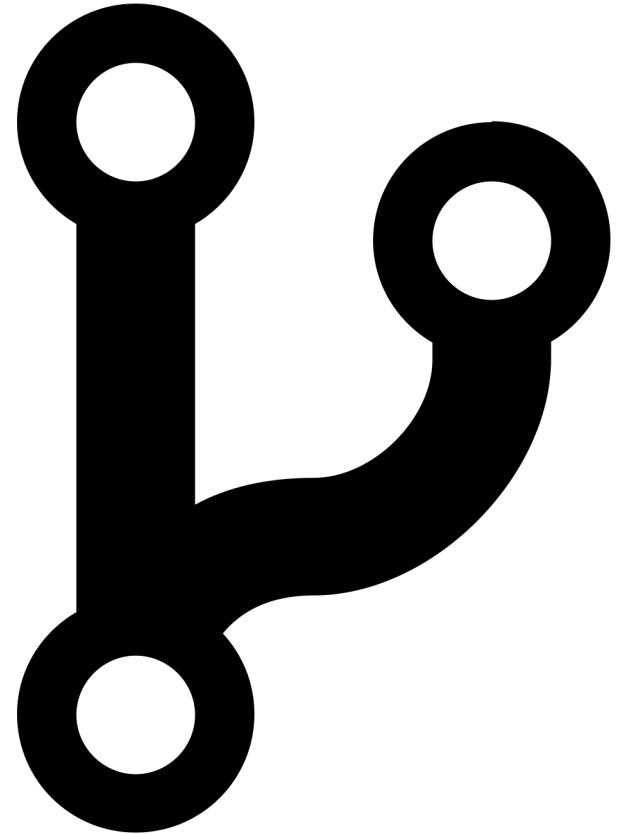
Utiliser WinMerge : la toolbar



Sauvegarde
après résolution

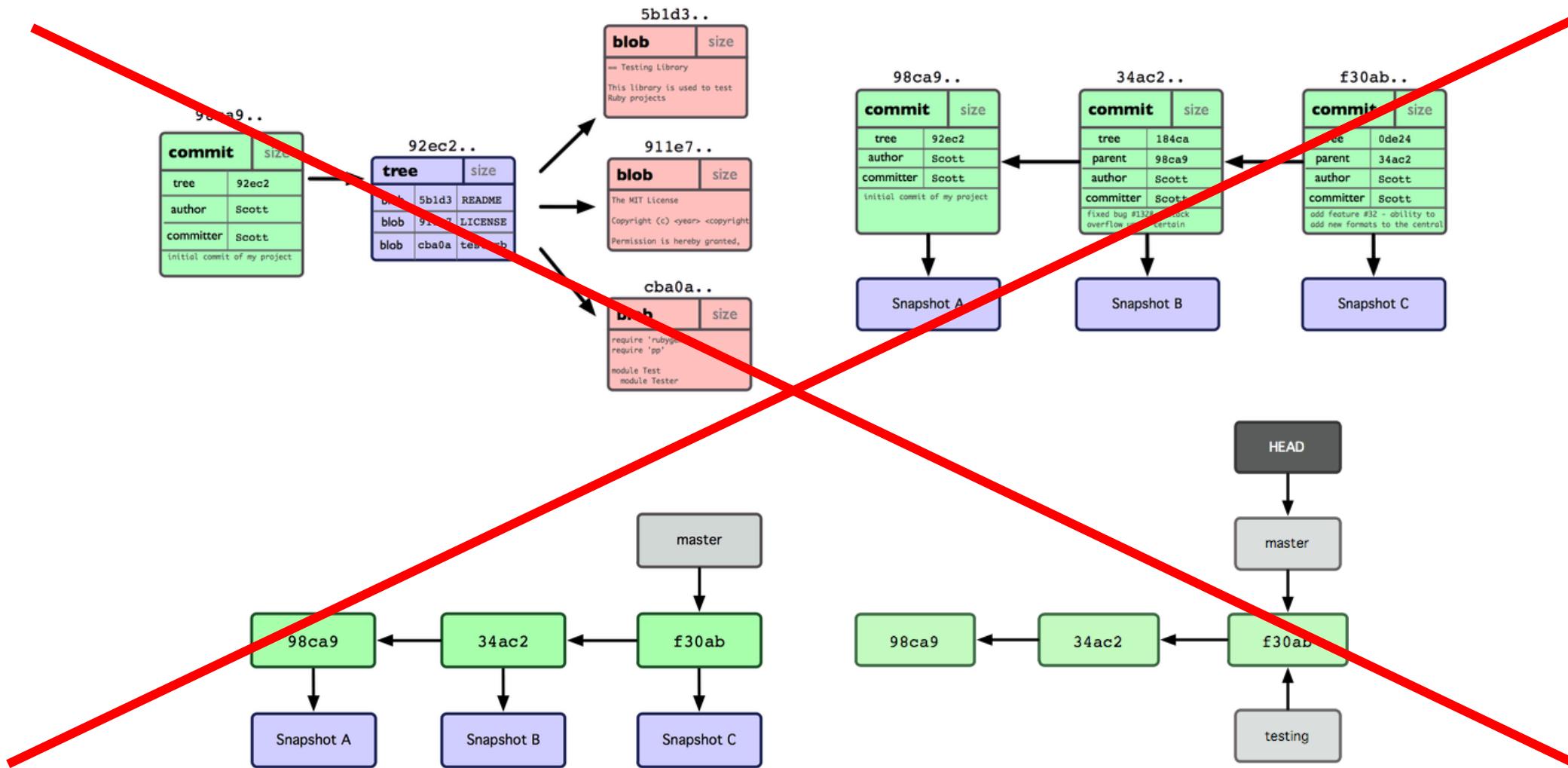
Différence
suivante/précédente

Utiliser le contenu à gauche/droite
pour résoudre la différence

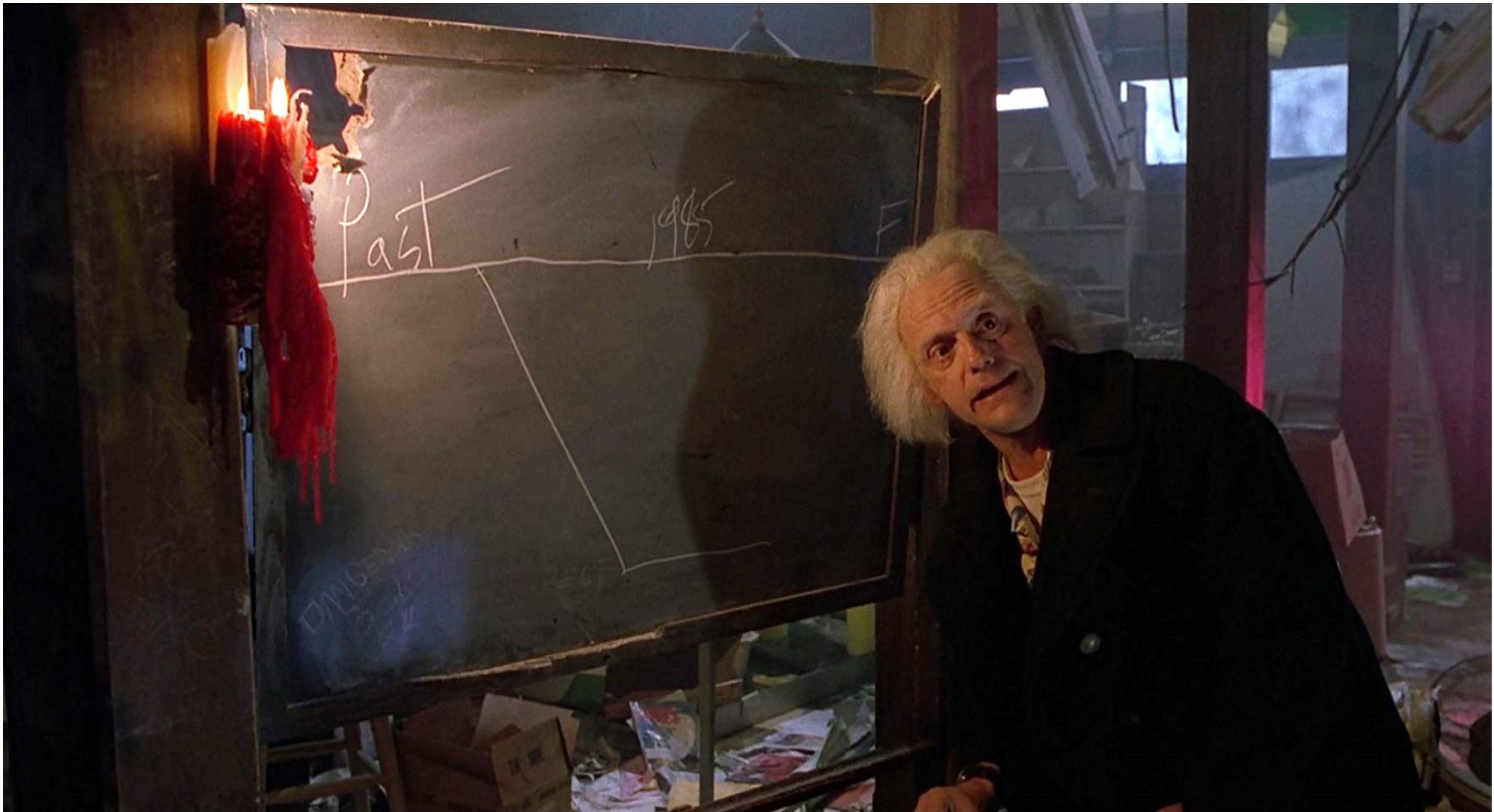


Git : les branches

Les branches

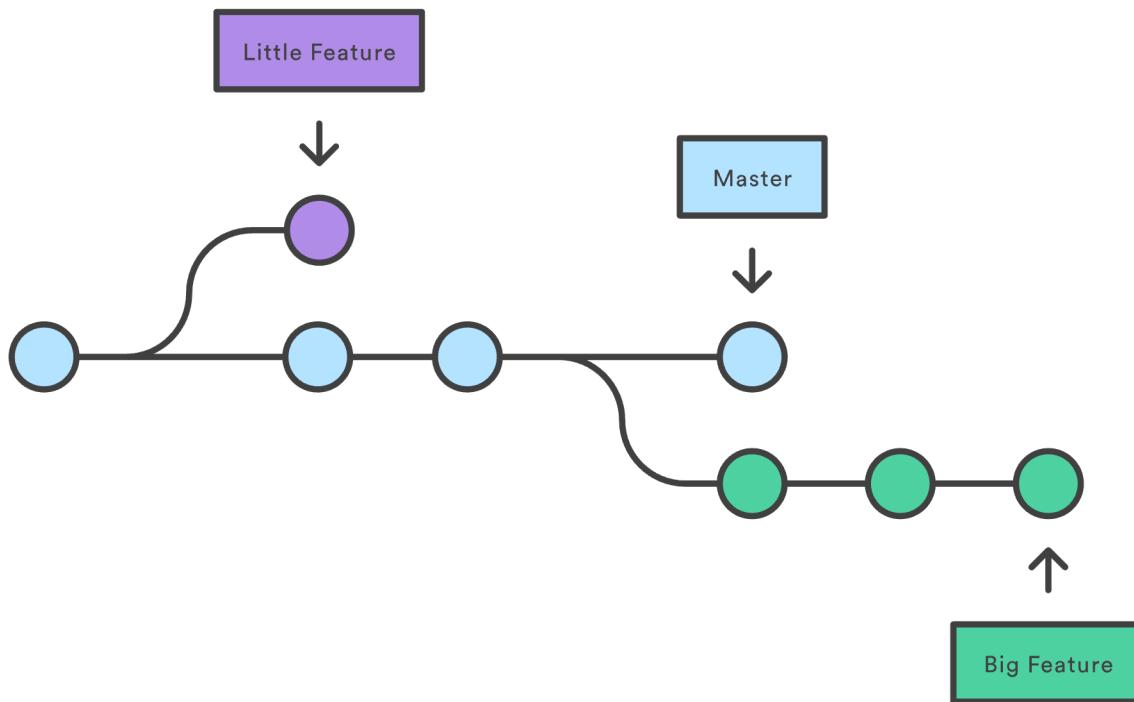


Les branches



Les branches

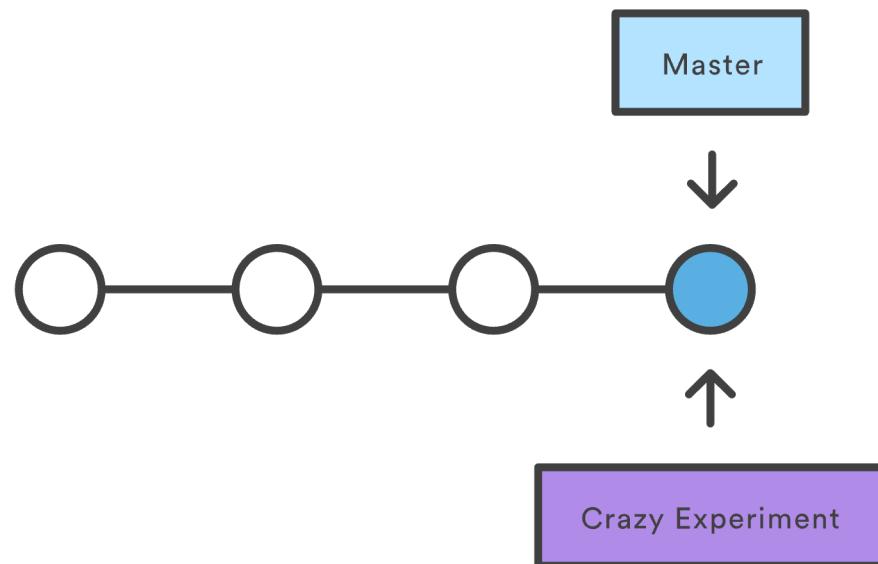
Branche = version alternative du code



Plusieurs objectifs

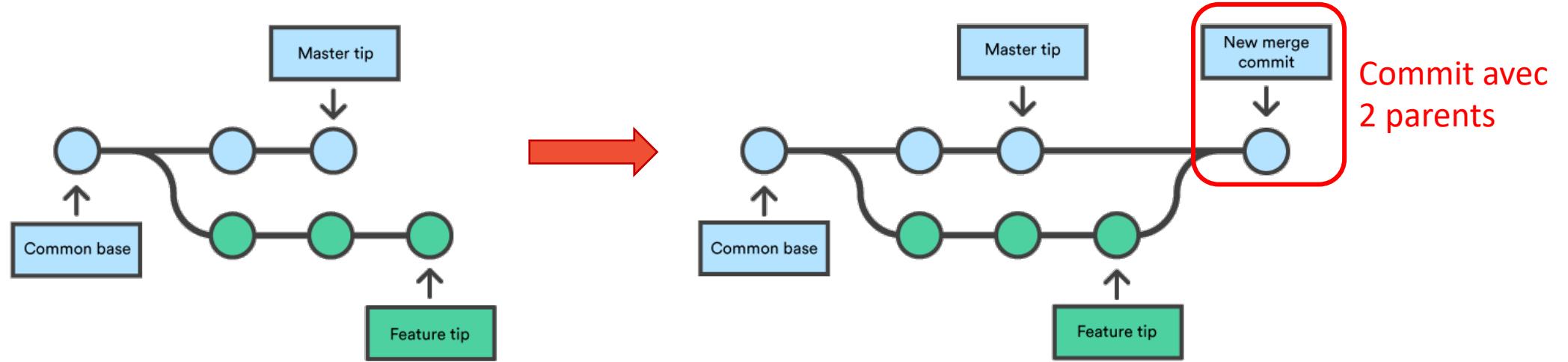
- Limiter les risques de pollution entre les variantes
 - Travailler à plusieurs sur différents sujets
 - On peut fusionner les branches entre elles => merge
 - Implémentation légère : git pointe vers 1 commit de référence

Les branches



2 branches pointent sur le
même commit ?
=> Importance de la HEAD

Merge

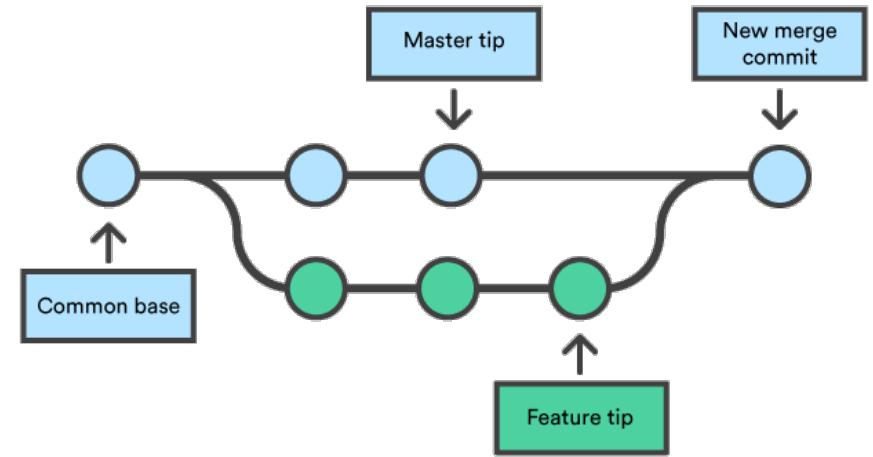


- Fusion de l'historique des commits
- On merge 1 branche dans la branche actuelle (merge into)

Schema by Atlassian is licensed under CC BY 2.5

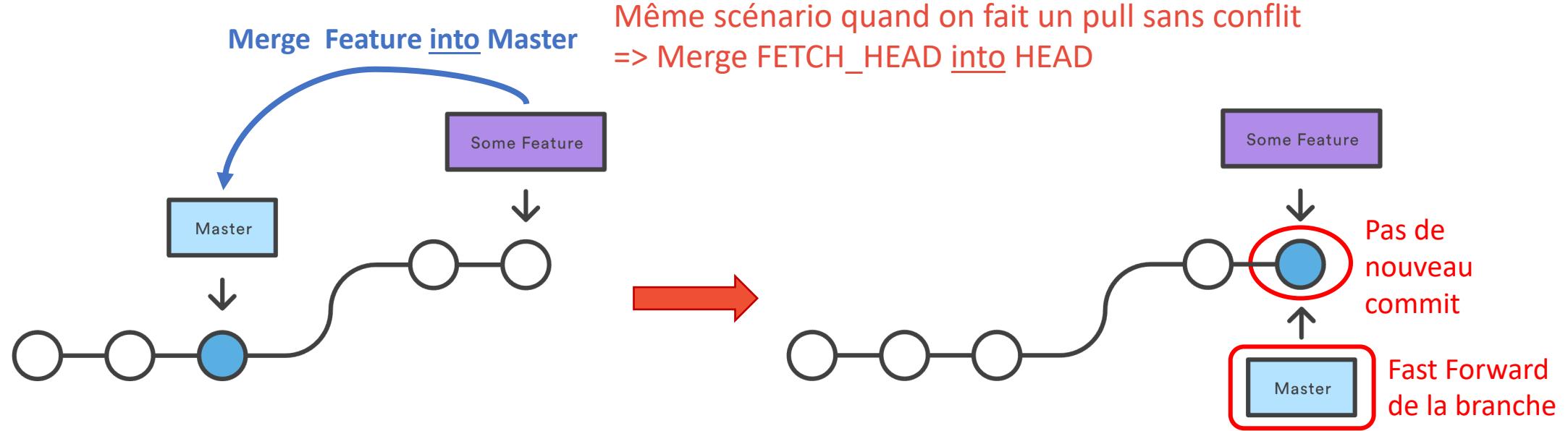
Merge

- Il faut avoir ses 2 branches à jour en local
- Historique des *commits* est préservée
- Git créé un nouveau *commit*
- Tentative de *merge* auto par git, sinon conflit à gérer



Schema by Atlassian is licensed under CC BY 2.5

Merge FF (Fast Forward)



Si historique linéaire entre les branches => Fast Forward possible

Ne pas créer un commit de merge n'est pas toujours une bonne chose !

Mise en pratique

Exercice

1. Se mettre en binôme
2. Créer un repo sur GitHub
3. Inviter le binôme sur le repo (settings -> manage access) en RW
4. Cloner le projet et faire le initial commit (par le proprio du repo)
5. Le binôme peut cloner ensuite
6. Créer un fichier txt à son nom (ex : \${LASTNAME}.txt)
7. Modifier son fichier
8. Commit
9. Pull => potentiellement, commit de merge automatique
10. Push
11. Répétez les étapes 6 à 9 pour être à l'aise avec la procédure

Mise en pratique

Exercice

1. Reprendre les précédents binômes
2. Inverser les rôles de créateur/invité
3. Créer un fichier readme.txt puis Commit + Push (par le proprio)
4. Faire un pull du côté du binôme
5. Modifier le même fichier et faire des commits des 2 cotés, pour avoir des conflits à tour de rôle ☺
 1. User1 modifie, commit et push
 2. User2 ne pull PAS
 3. User2 fait une modif et commit
 4. User2 pull => conflit !
6. Résoudre les conflits



git : concepts utiles

Benoit Verdier - EPITA

Git Stash

- Permet de mettre de coté son travail sans faire de commit
- Git stash génère un diff qu'on peut re-appliquer + tard
- Intérêt : mettre de coté ses modifs pour changer de branche
- ⚠ Fichier stocké en local sur l'ordi, pas sur le serveur ⚠

Git ignore

- Très important : ne pas versionner les fichiers inutiles
- À initialiser en début de projet
- Fichier `.gitignore`

Git ignore : quoi ignorer ?

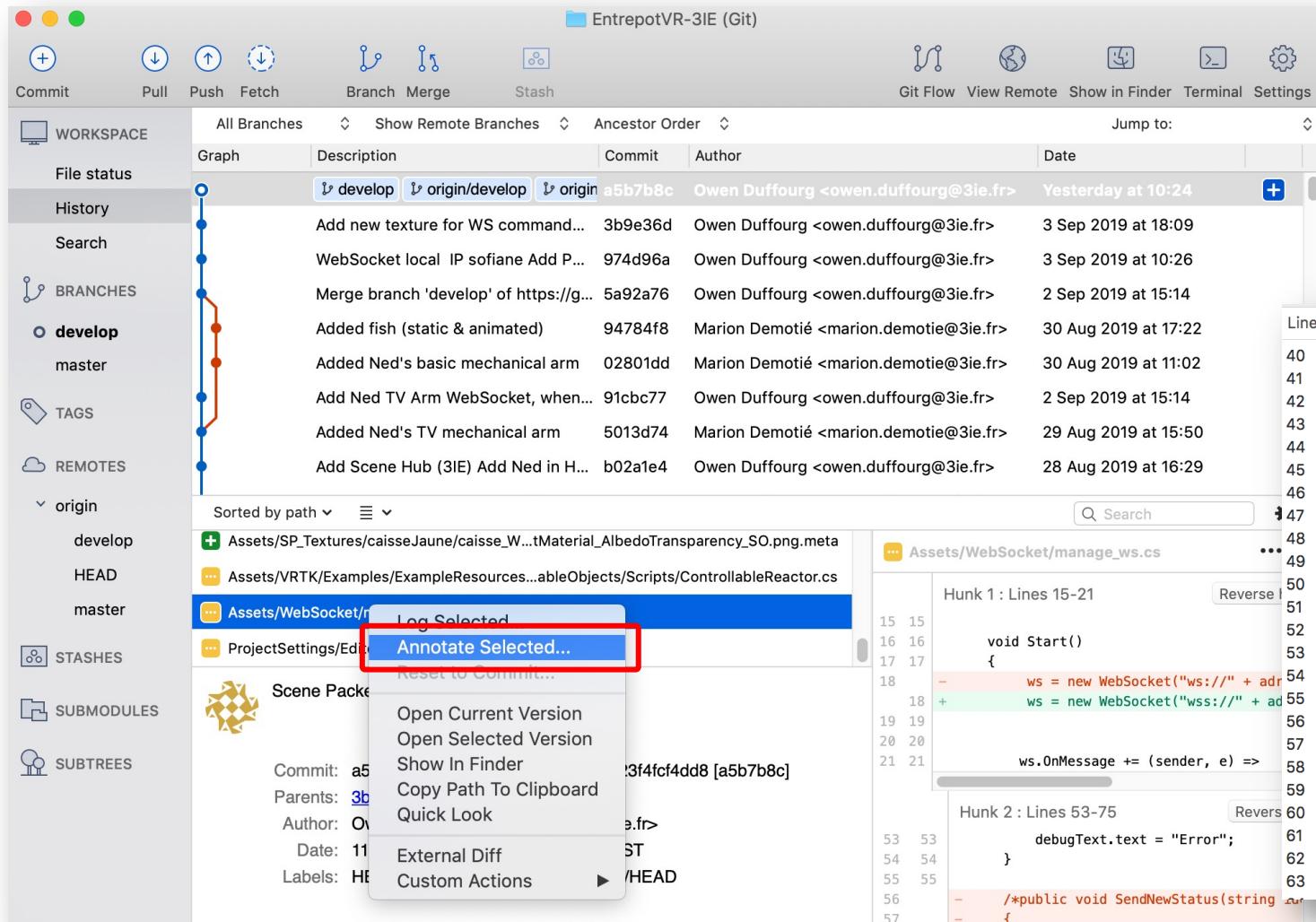
3 grandes catégories à ignorer

- Ce qui est lié à l'OS et aux IDE
 - Pour l'IDE : préférences propres au user, ...
 - Par l'OS : vignettes d'images, fichiers pour l'indexation, ...
- Ce qui est généré
 - Artefacts de compilation
 - Fichiers liés à l'exécution
- Ce qui est téléchargé (ex : fichiers package manager)

Git ignore pertinent

- Ne pas refaire les refaire de zéro à chaque
- Bonne base : <https://github.com/github/gitignore>
- ex iOS : on peut combiner
 - <https://github.com/github/gitignore/blob/master/Swift.gitignore>
 - <https://github.com/github/gitignore/blob/master/Global OSX.gitignore>
- Pensez aux gens sous Mac, mettez toujours le OSX ignore :)

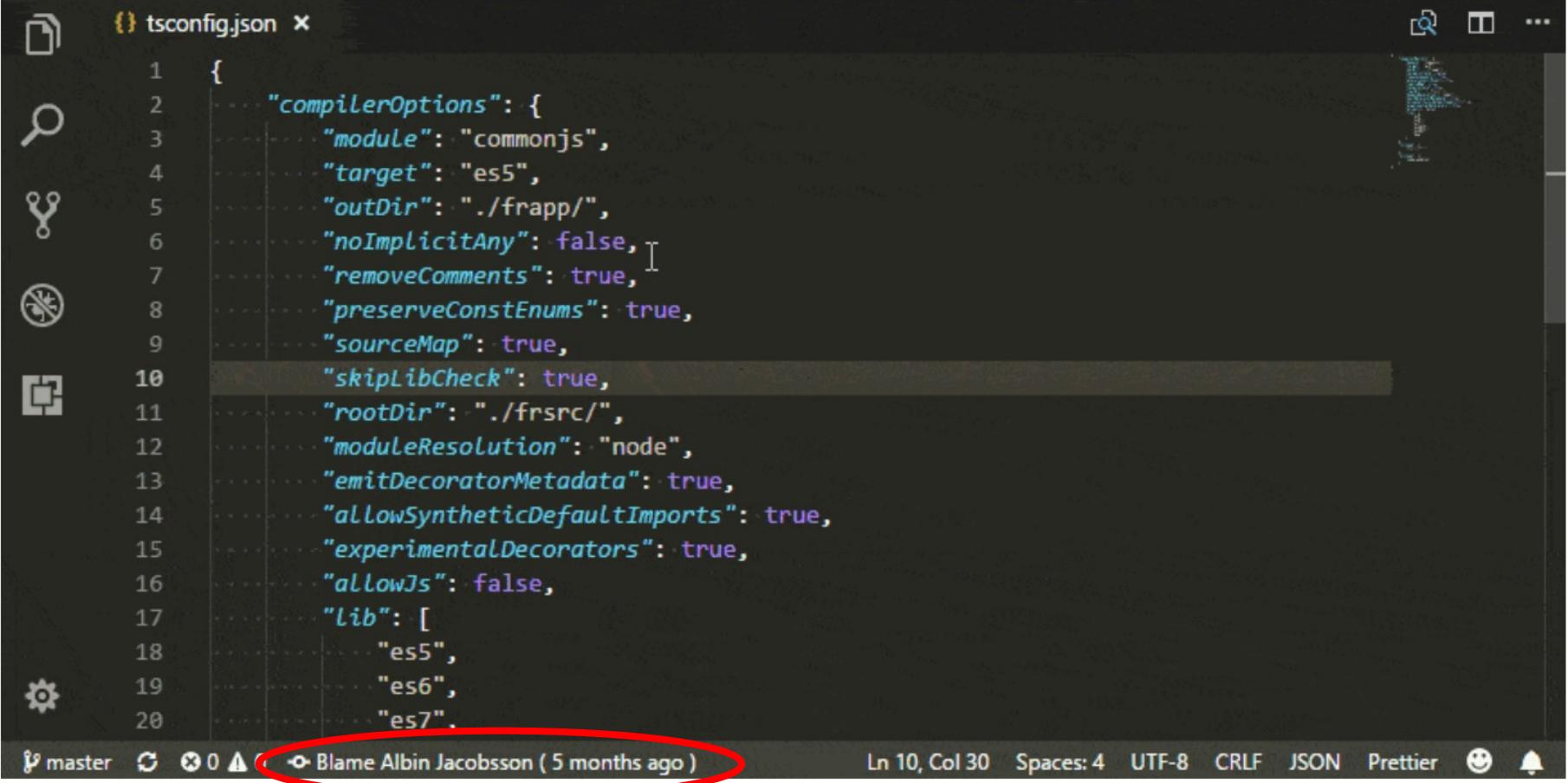
Git blame



Affiche les auteurs ligne par ligne
Aussi appelé annotate ou annotation

Git blame

Intégration dans
les IDE aussi

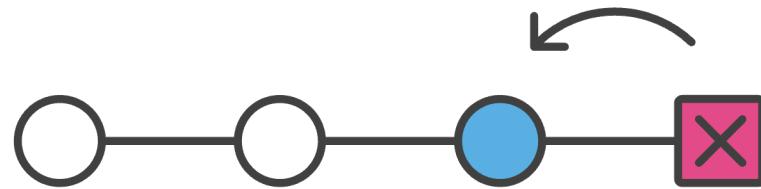


A screenshot of a code editor window showing the `tsconfig.json` file. The file contains configuration options for TypeScript compilation. A red oval highlights the status bar at the bottom of the editor, which displays the message "Blame Albin Jacobsson (5 months ago)".

```
{}
tsconfig.json x
1  {
2    "compilerOptions": {
3      "module": "commonjs",
4      "target": "es5",
5      "outDir": "./frapp/",
6      "noImplicitAny": false,
7      "removeComments": true,
8      "preserveConstEnums": true,
9      "sourceMap": true,
10     "skipLibCheck": true,
11     "rootDir": "./frsrc/",
12     "moduleResolution": "node",
13     "emitDecoratorMetadata": true,
14     "allowSyntheticDefaultImports": true,
15     "experimentalDecorators": true,
16     "allowJs": false,
17     "Lib": [
18       "es5",
19       "es6",
20       "es7".
```

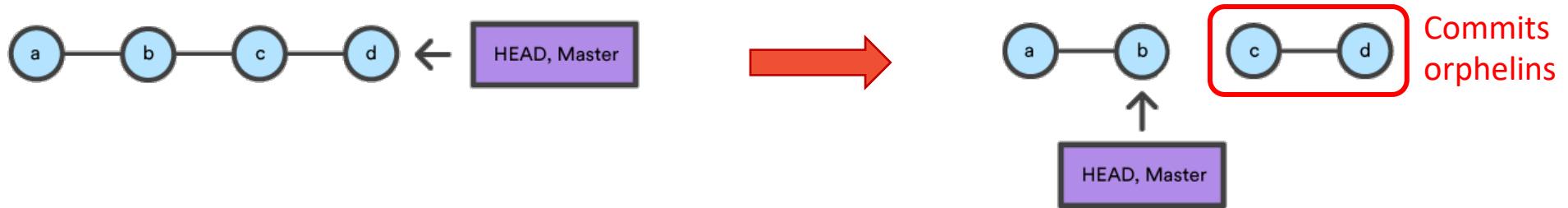
master 0 Blame Albin Jacobsson (5 months ago) Ln 10, Col 30 Spaces: 4 UTF-8 CRLF JSON Prettier

Git reset commit



- Permet de *reset* à un commit précis

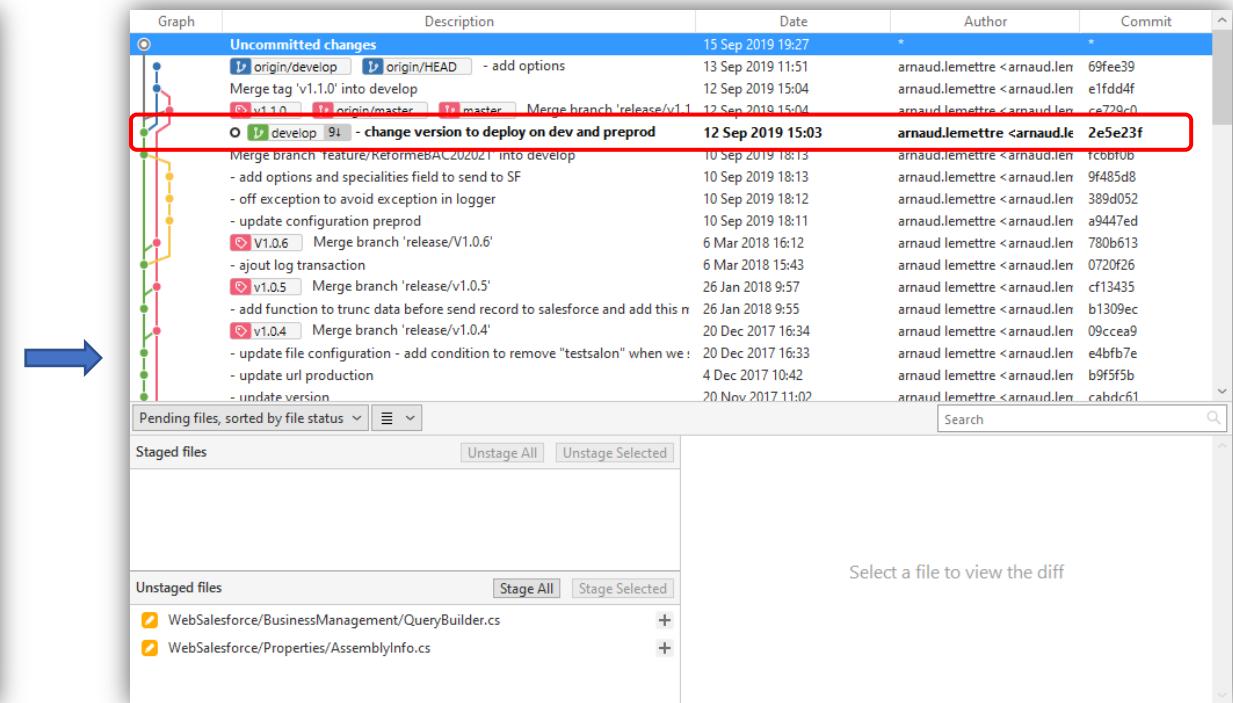
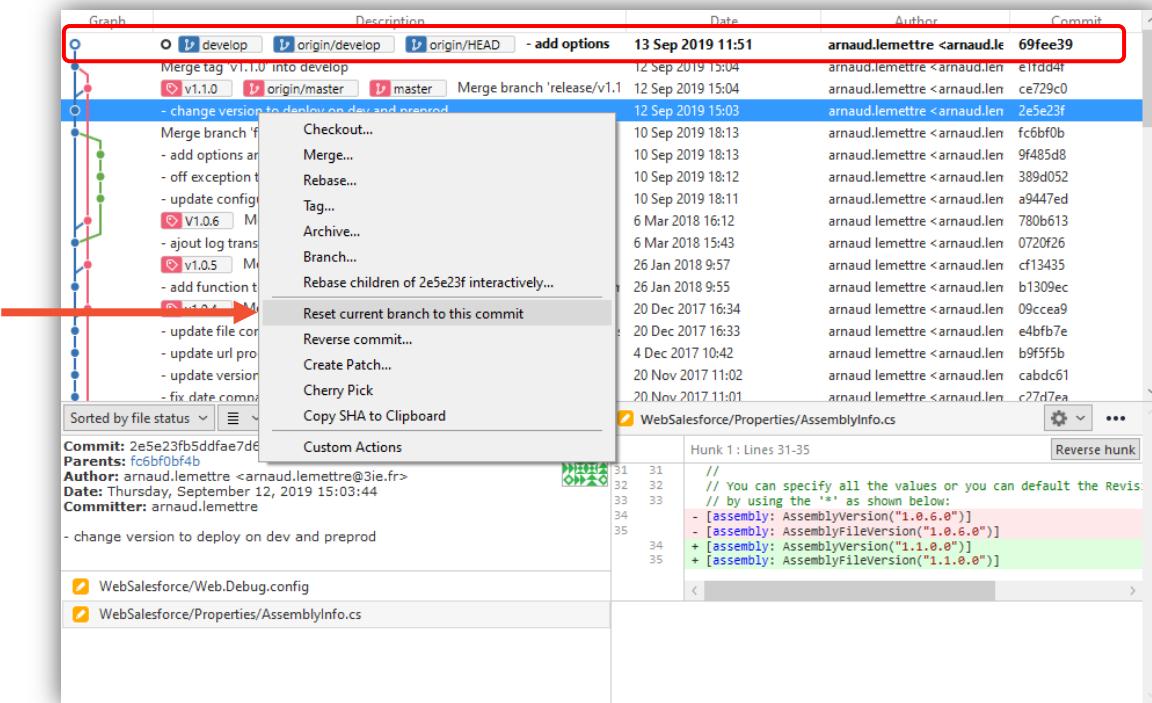
Git reset commit



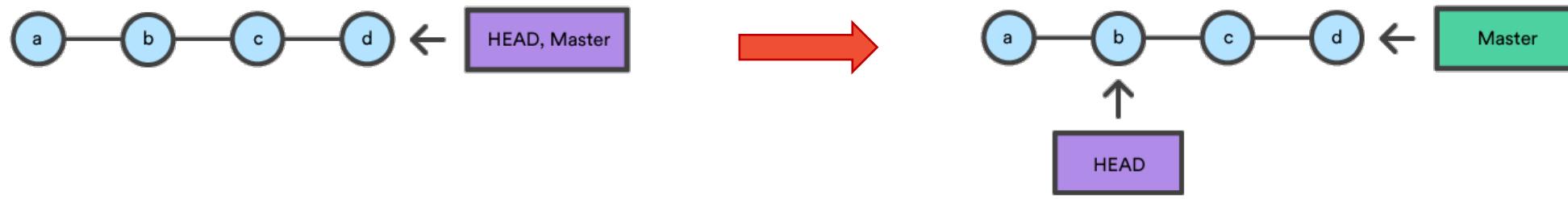
- On déplace la *reference* de la branche
=> *HEAD* reste sur 1 nom de branche mais *refs/\$BRANCH* change
- Si branche locale, on perd les commits orphelin
- Si branche remote, on ne peut pas push (sauf force push)

Schema by Atlassian is licensed under CC BY 2.5

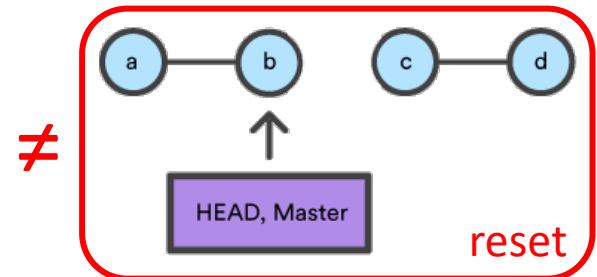
Git reset en GUI



Git checkout commit



- Restaure les fichiers d'un commit
- On ne déplace que la *HEAD*
- *HEAD* référence un *hash* de commit
- ~~branche~~ => *detached HEAD*



Schema by Atlassian is licensed under CC BY 2.5

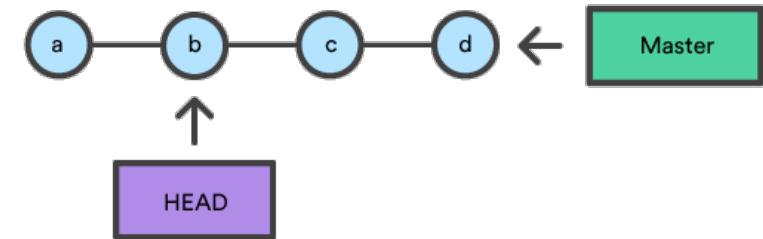
Git checkout commit = detached Head

⚠️ Dangereux ⚠️

Ca ne se manipule pas comme
une branche (ex : pas de *merge*)

Utile pour :

- Récupérer une version précise
- Certaines fonction avancées



Schema by Atlassian is licensed under CC BY 2.5

Git tag

- Permet d'étiqueter le repo (ex: v1.0.4)
- Simple référence vers un commit => c'est performant

Le tag peut stocker

- Nom du tag
- Nom du tagger
- Message



C'est utilisé sur GitHub pour les release

Le force push

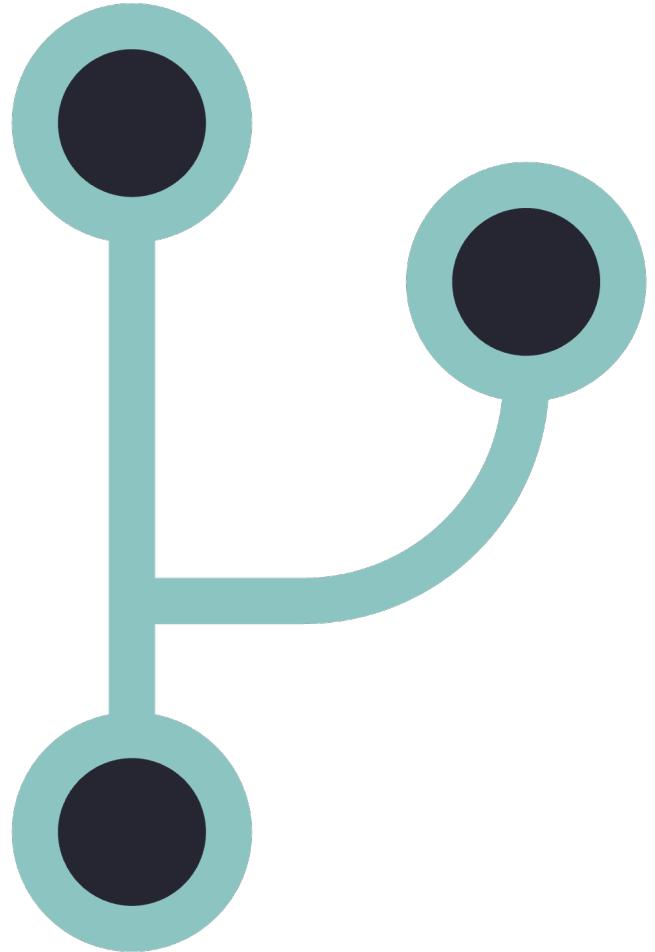
⚠ Interdit ⚠

Force la main au server et écrase les commits est présent

Met en conflit le git des autres dev

Utilité très spécifique

- On veut modifier (*to amend*) le dernier commit
- On veut vraiment ré-écrire une branche



Le git-flow

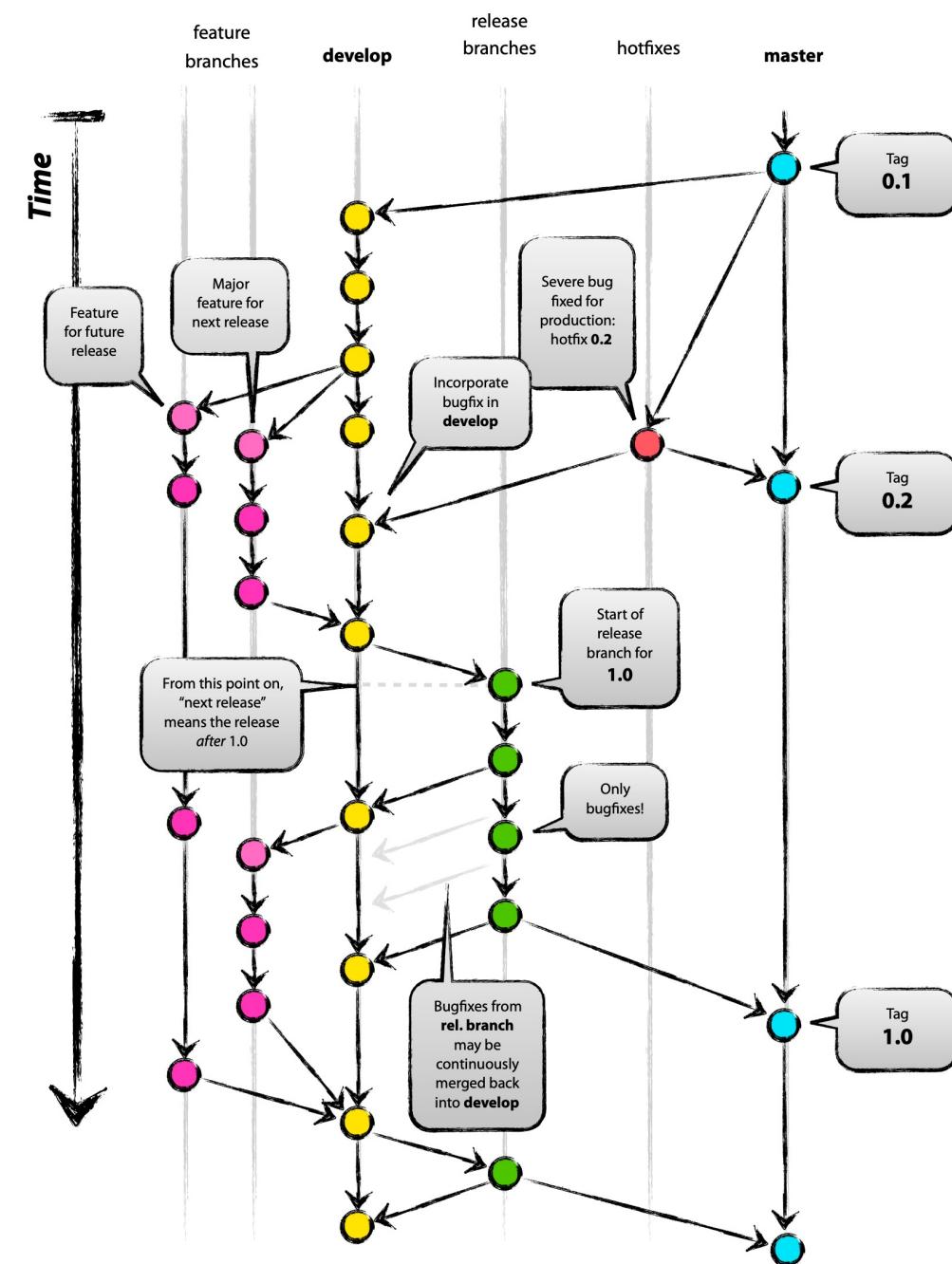
Benoit Verdier - EPITA

Git-flow

- Implémentation des concepts de « branching » décrit par Vincent Driessen
- **Donne un sens aux branches**
- **Formalise un processus**
- C'est une extension git => s'utilise sur chaque poste de dev

<https://nvie.com/posts/a-successful-git-branching-model/>

Git-flow : branching model

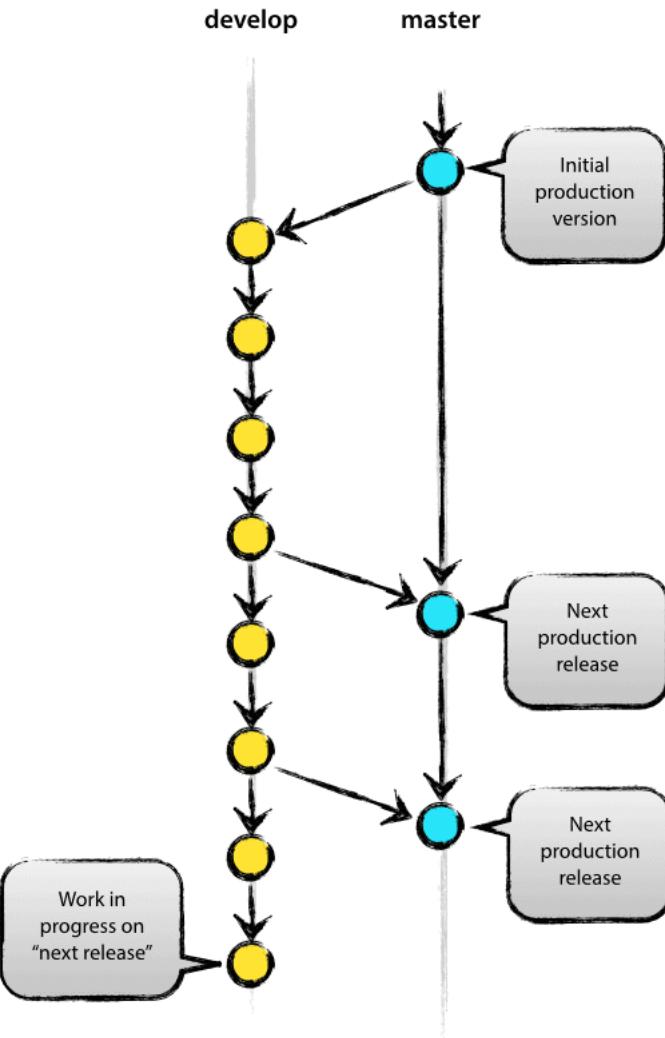


Git-flow : branches principales

- Branches à durée de vie infinie
- « develop » pour dev
- « master/main » pour le *production-ready*

On comprend l'idée mais comment :

- Travailler de manière cloisonnée ?
- Gérer ses releases ?



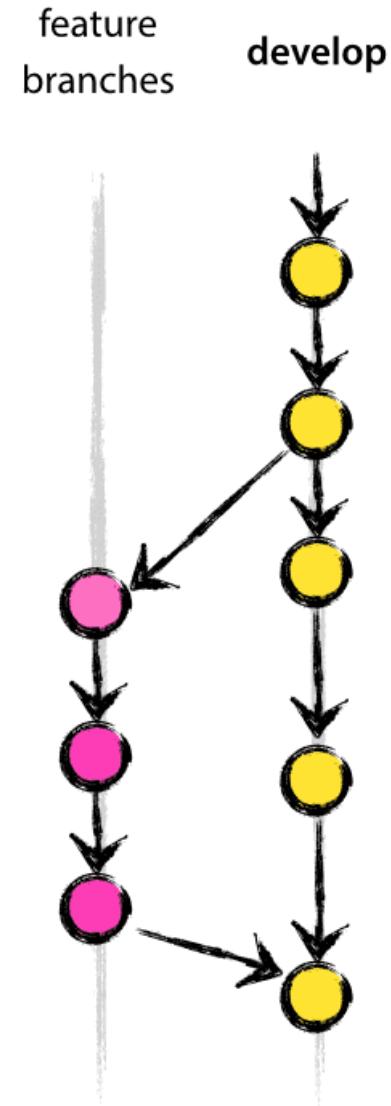
Git-flow : feature branch

Plusieurs usages

- Workflow centré sur les fonctionnalités
- Gros dev
- R&D

Quand la feature est finie

- On merge dans develop
- On supprime la branche



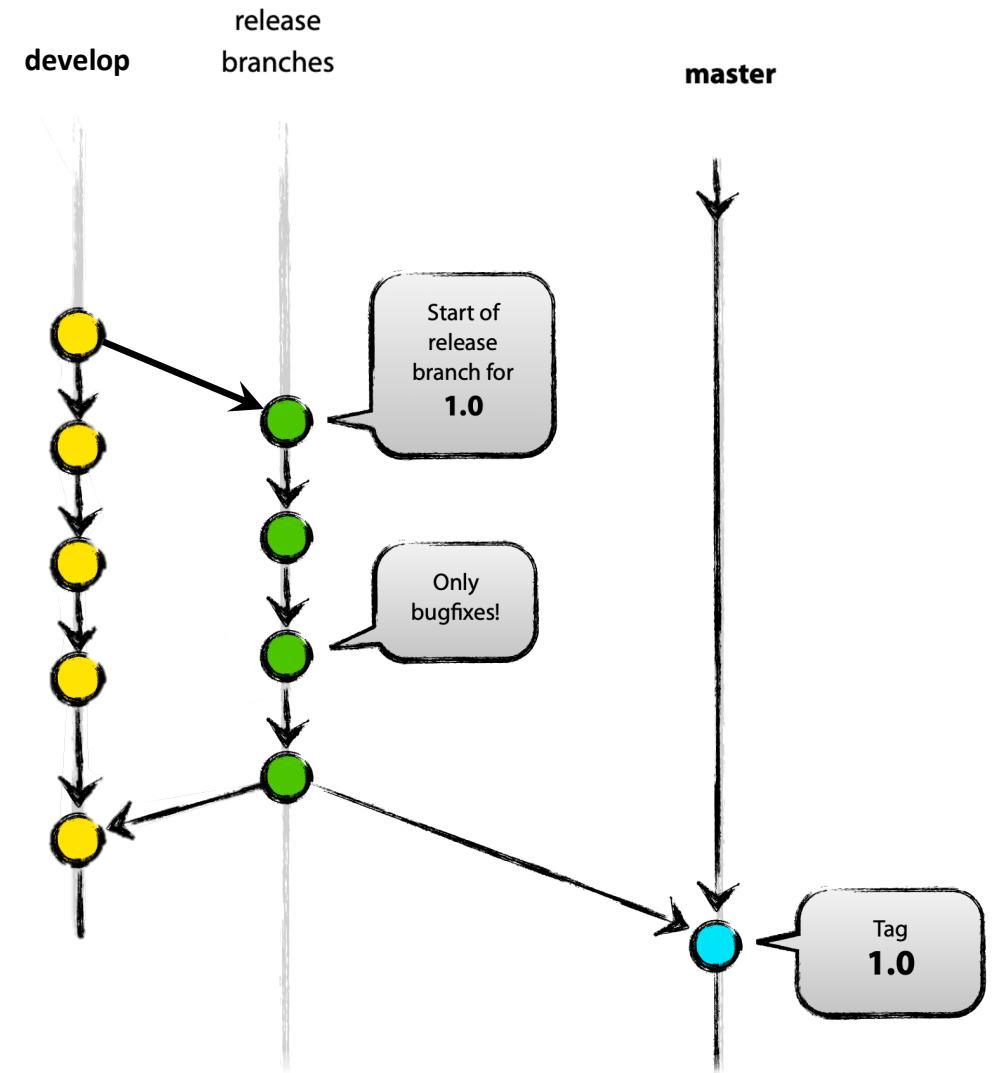
Git-flow : release branch

Pour préparer les releases :

- On stabilise à partir de develop
- Pas de nouvelles features
- Juste des correctifs

Une fois release validée :

- Merge dans master et develop
- Tag
- Delete de la branche



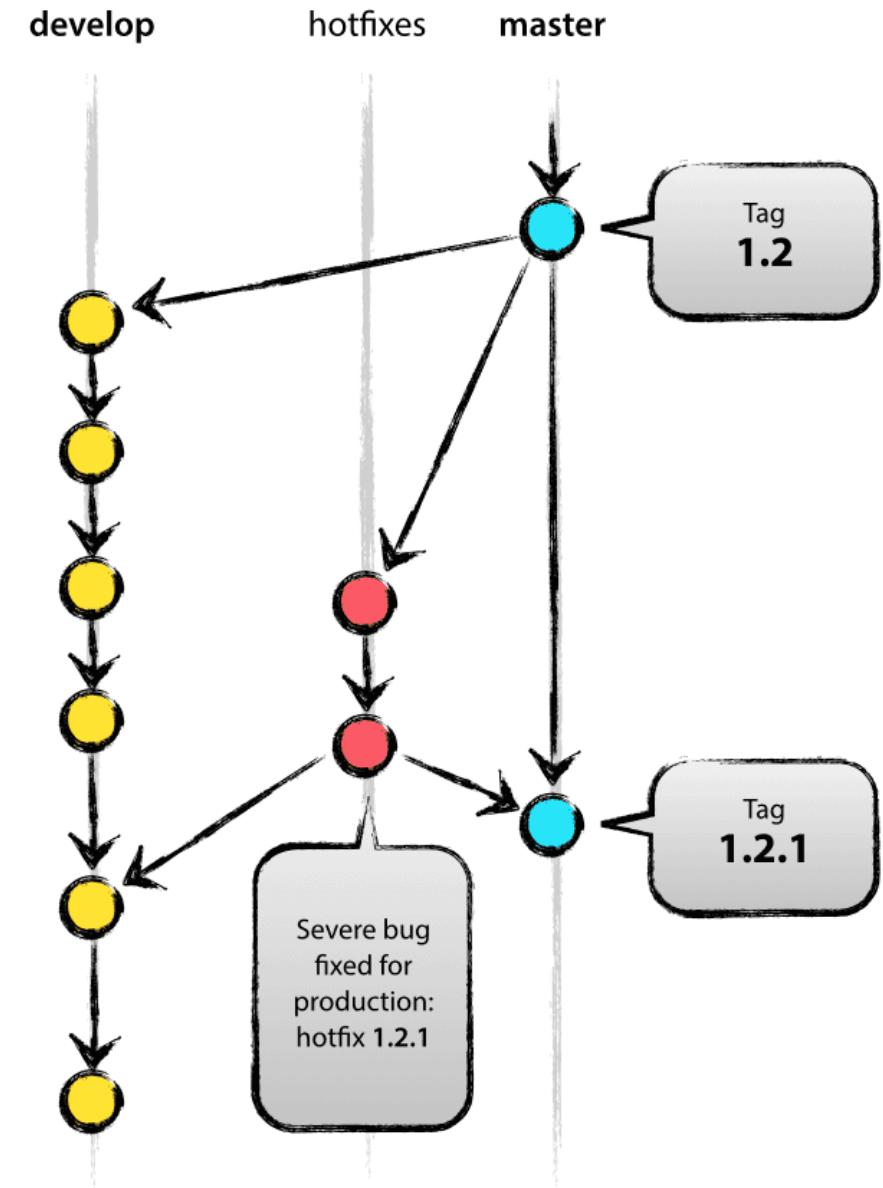
Git-flow : hotfix branch

Pour corriger une précédente release

- Se fait à partir de master

Quand hotfix validé :

- Merge dans master et develop
- Tag
- Delete de la branche



Git-flow : Que fait l'extension ?

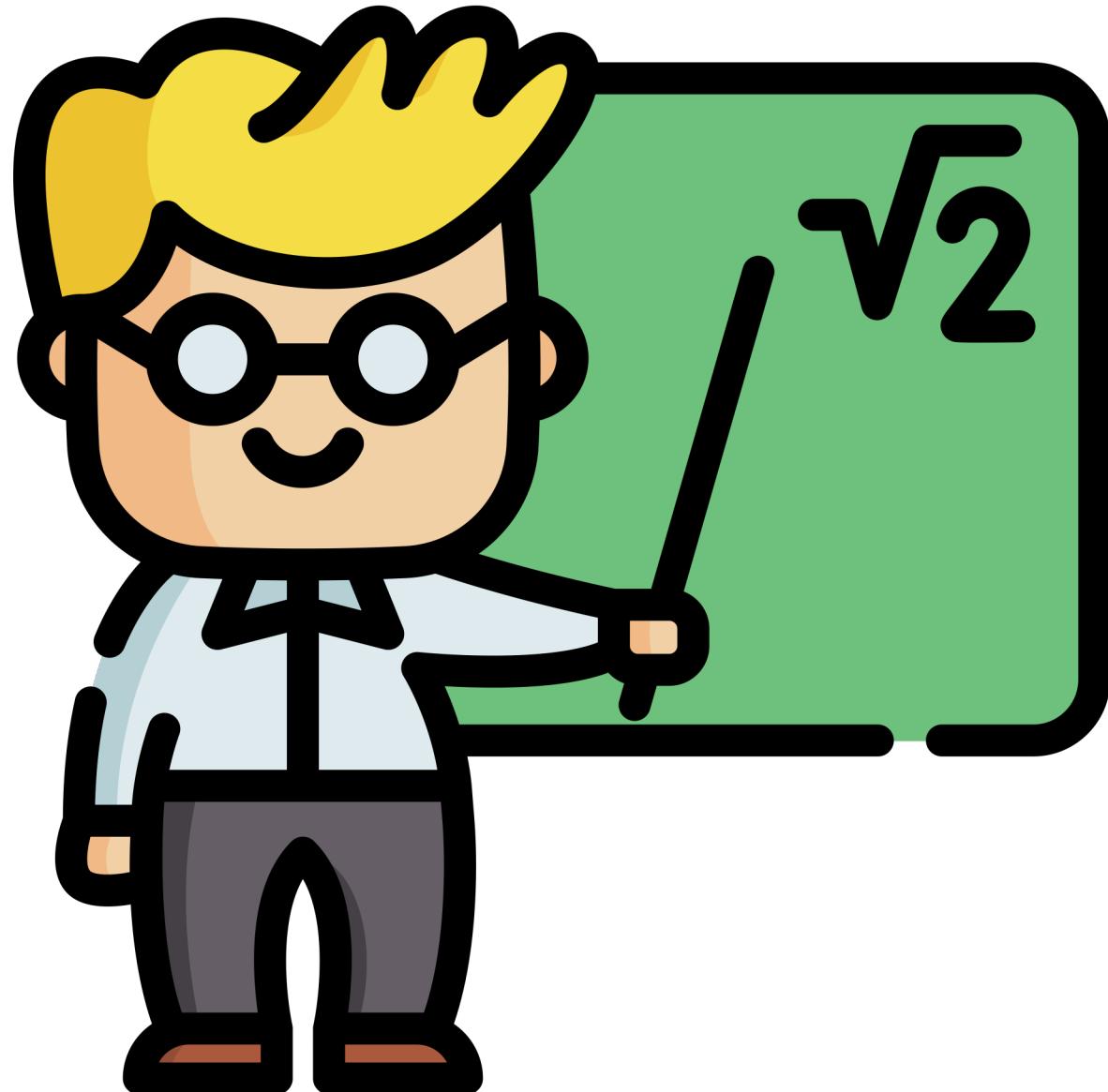
Nommage de branches

Automatisation des manipulations (tag, merge)

Sélections des branches suivant les actions

- ! Le dev doit bien delete ses branches !

Démo
git-flow



Mise en pratique

Exercice part 1

1. Créez un repo git depuis votre Github et clonez le
2. [main] Créez un readme.md (puis Commit + Push)
3. Initialisez le git-flow depuis SourceTree
4. [develop] Modifiez votre readme.md (puis Commit)
5. [develop] Utilisez le reset pour annuler votre commit
6. [develop] Faites un discard de modification sur le fichier readme.md

Exercice part 2

On garde le même repo que précédemment

1. Créez une feature « todo » depuis le git-flow
2. [feature/ todo] Créez un fichier courses.txt avec « PQ et farine » par ex (puis Commit + Push)
3. [develop] Mettez à jour votre readme.md (puis Commit + Push)
4. [feature/todo] Terminez votre feature, ce qui doit déclencher un merge de **todo** into **develop**
5. Finalisez le commit de merge (puis Push)

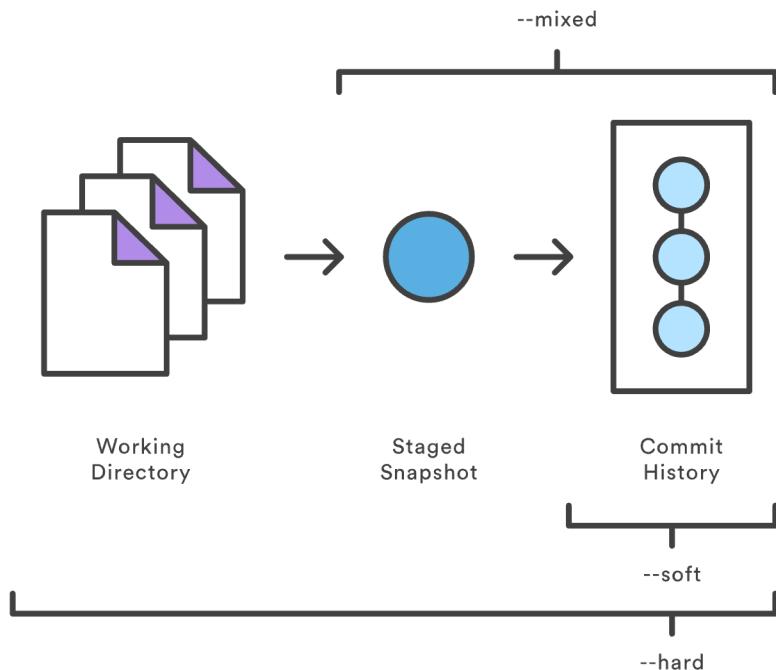
Exercice part 3

On garde le même repo que précédemment

1. [develop] Créez une release « v1.0 » depuis le git-flow
2. [release/v1.0] Corrigez le fichier « courses.txt » : -PQ +Papier toilette
(puis Commit + Push)
3. [release/v1.0] Terminez votre release (puis Push)
4. Vérifiez que le git-flow a bien merge dans develop et main
5. Vérifiez que vous avez bien le tag « v1.0 »

Let's rewrite
history

Git reset



--soft

Does not touch the index file or the working tree at all (but resets the head to <commit>, just like all modes do). This leaves all your changed files "Changes to be committed", as git status would put it.

--mixed

Resets the index but not the working tree (i.e., the changed files are preserved but not marked for commit) and reports what has not been updated. This is the default action.

--hard

Resets the index and working tree. Any changes to tracked files in the working tree since <commit> are discarded.

<https://www.atlassian.com/git/tutorials/undoing-changes/git-reset>

Schema by Atlassian is licensed under CC BY 2.5

Git reset : en pratique

HEAD toujours sur le nom de la branche

Ref pointe sur un vieux commit

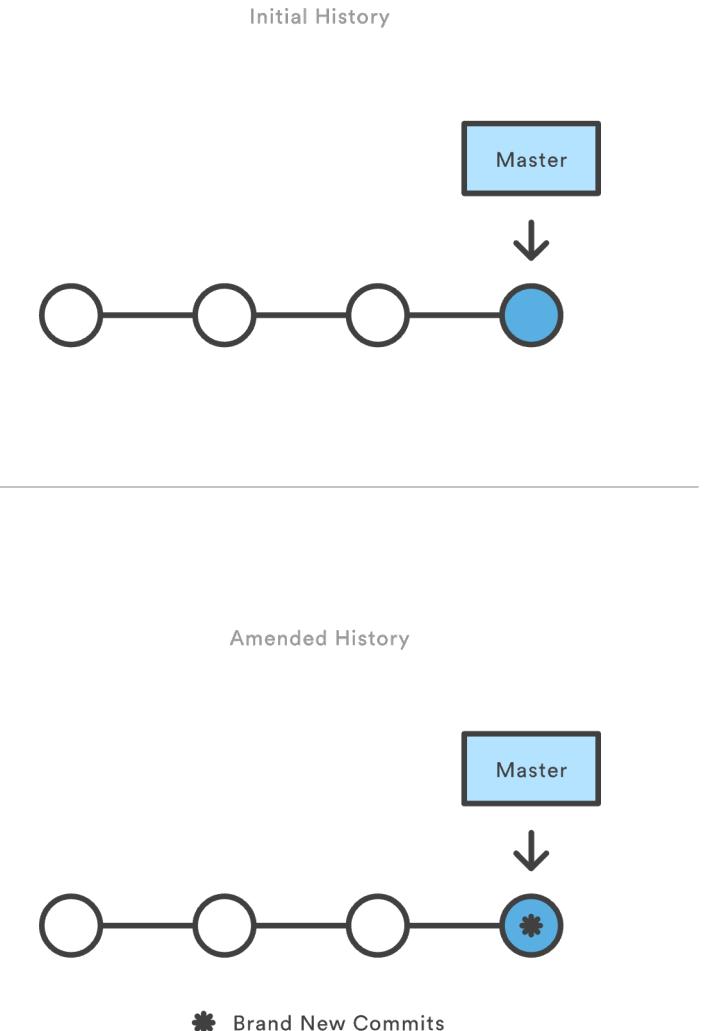
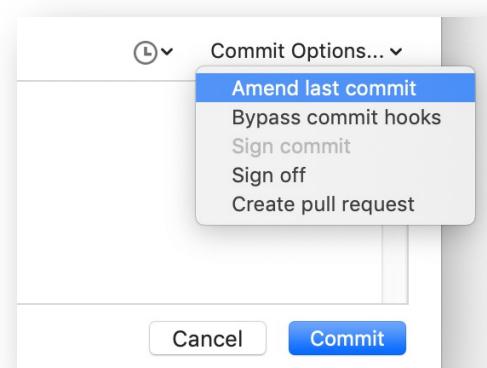
Reset	<i>Reseted Commits</i>	Contenu concerné par les <i>reseted commits</i>
Soft	Perdus	Dans l' <i>index</i>
Mixed	Perdus	Dans le <i>working dir</i>
Hard	Perdus	Perdus

Commit orphelin qui vont être
garbage collected

Amending a commit

- Pour modifier le dernier commit (non push)
- Utilité :
 - Changer le « commit message »
 - Ajouter des modifications

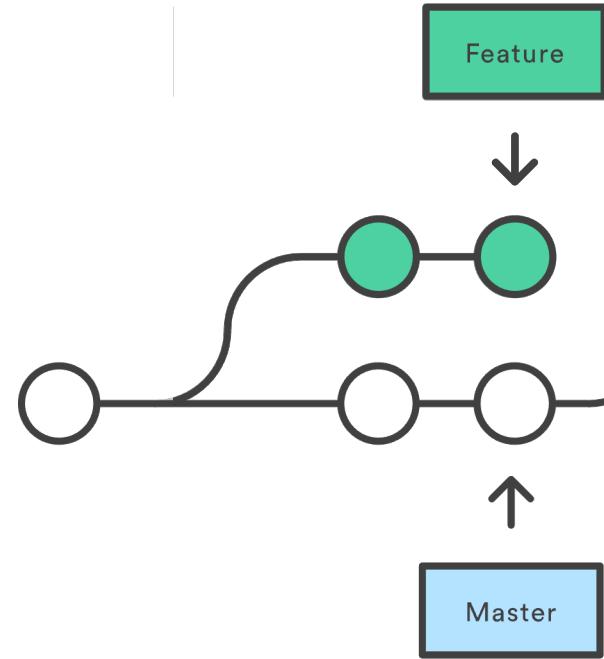
Avec Source Tree :
Option lors du commit



Schema by [Atlassian](#) is licensed under CC BY 2.5

Le rebase

« Rebasing is the process of moving or combining a sequence of commits to a new base commit »



Schema by [Atlassian](#) is licensed under CC BY 2.5

Le rebase

- Rebase : parce qu'on change la base
- On parle aussi de « rewriting history » ou « replaying commits »
- Utilité :
 - Garder un historique linéaire
 - Retravailler des commits

⚠ Pas si on déjà push ⚠

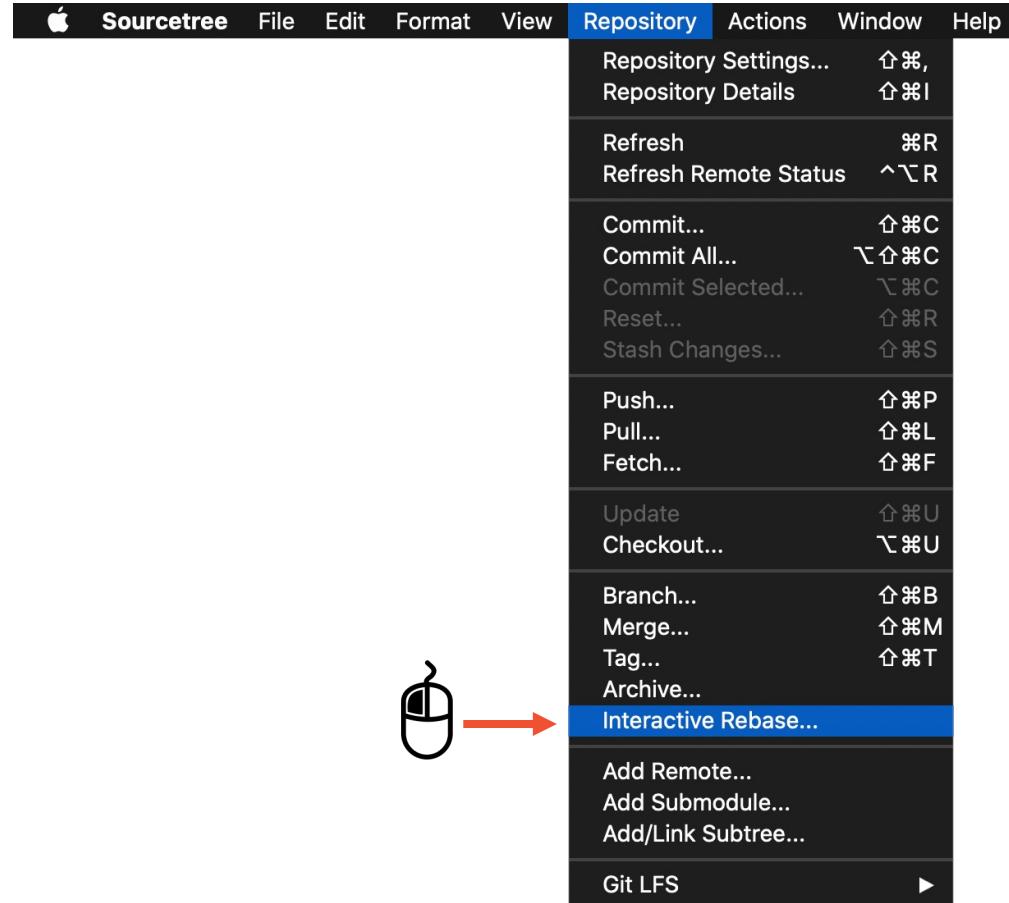
⚠ On manipule des nouveaux push ⚠

Interactive rebase : utilité

Pour retravailler (interactivement) les commits avant de push

- Retravailler le msg de commit
- Fusionner des micro-commits
- Supprimer un commit
- Ré-agencer les commits
- Insérer un nouveau commit dans l'historique

Interactive rebase : UI





Interactive rebase : UI

« Squash » = Ecraser

Reorder and amend commits:

Changeset	Amend Commit?	Description	Commit Date
a06026a	<input type="checkbox"/>	Add imagesloaded to start building a solution for loading images	17 Apr 2013 07:04
b42db55	<input type="checkbox"/>	Add images loaded to start building a solution for loading images	17 Apr 2013 07:00
48c2c13	<input type="checkbox"/>	pagination and images loaded to start building a solution for loading images	12 Mar 2013 17:44

Edit message...
Squash with previous commit...
Delete commit...

 Add imagesloaded js from <https://github.com/desandro/imagesloaded> to start building a solution for loading images

Commit: a06026a2c3411afca4cc06874926faf2e5...
Parents: [b42db55880](#)
Author: Anton Evers <canton.evers@kega.nl>
Date: April 17, 2013 at 7:04:07 AM PDT
Labels: HEAD -> master origin/master origin/HE...

demo.html

Hunk 1 : Lines 13-19

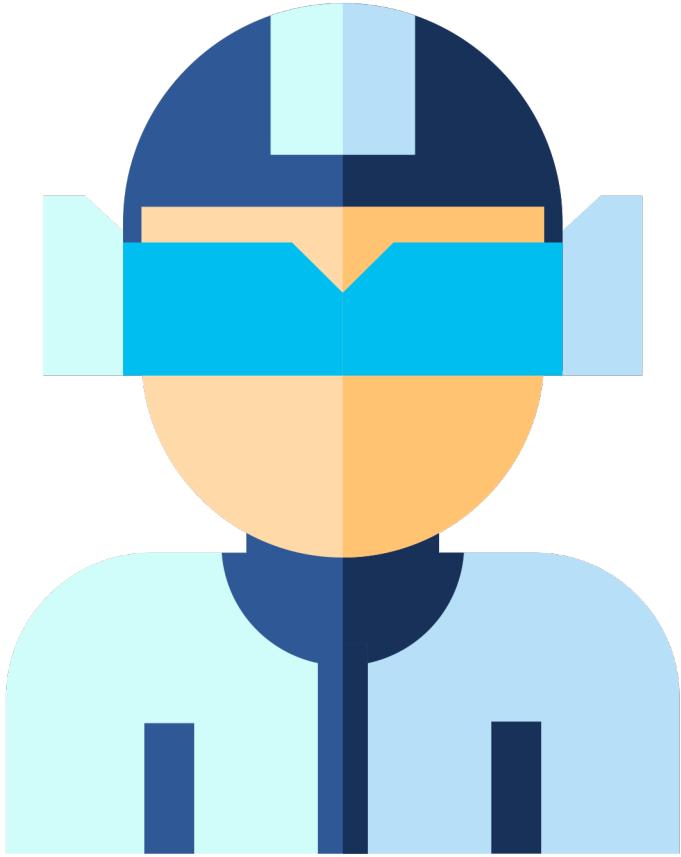
13	13	<!-- lib -->
14	14	<script type="text/javascript" src="js/lib/modernizr.js">
15	15	<script type="text/javascript" src="js/lib/jquery.js">
16	17	+ <script type="text/javascript" src="js/lib/imagesloaded.js">
17	18	<script type="text/javascript" src="js/lib/jquery.lazyload.js">
18	19	<script type="text/javascript" src="js/lib/jquery.lazyload.prefs.js">

? | Filename | Path

demo.html

jquery.imagesloaded.js js/lib

Reset Edit message Squash with previous Delete ▲ ▼ Cancel OK



Pour aller plus
loin

Git LFS : Qu'est ce que c'est ?

- LFS = Large File Storage
- Répond au besoin de stocker des binaires (VS du texte)
- Extension git

<https://git-lfs.github.com/>

Git LFS : Comment ca marche ?

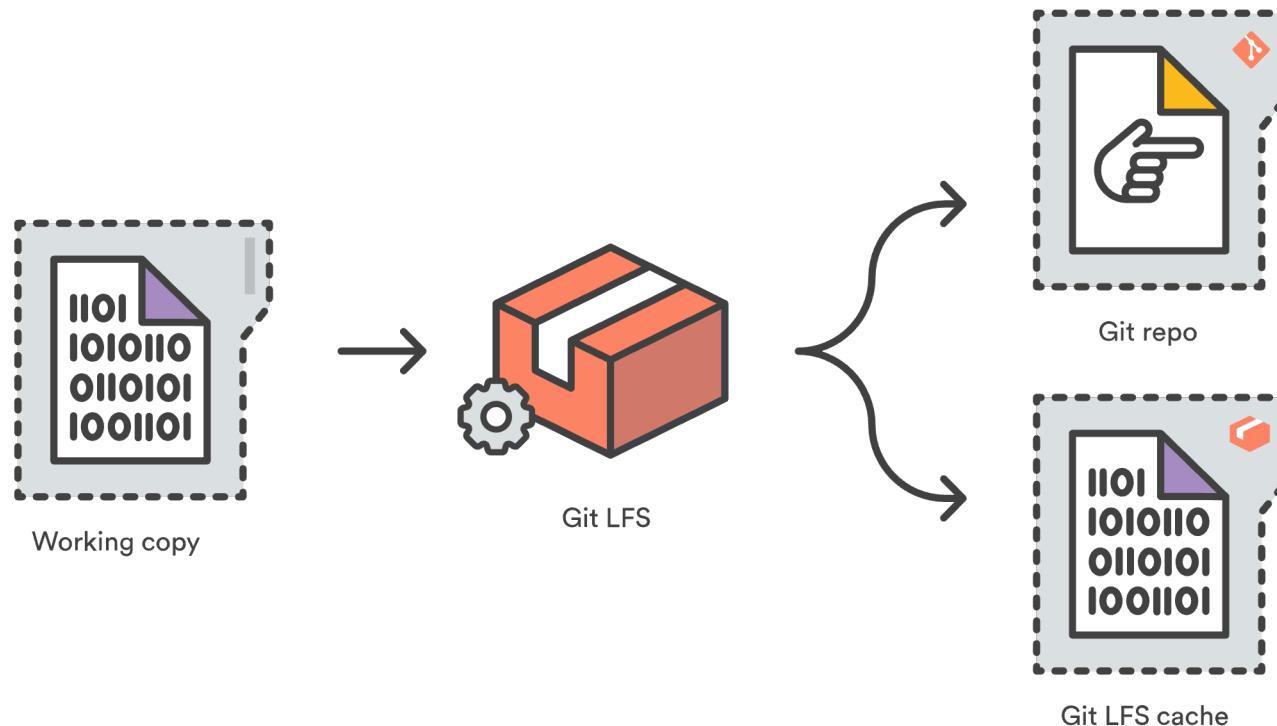


- Serveur git stocke des pointeurs vers le cache LFS
- Download transparent : extension aiguille les téléchargements
- Configuration avec pattern de nom de fichiers (ex: *.jpg)

<https://git-lfs.github.com/>

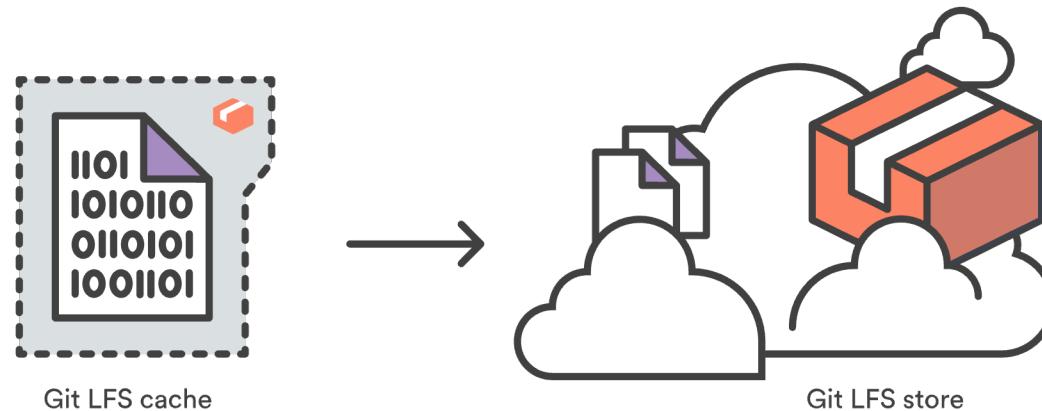
Git LFS

Step 1 : `git add`



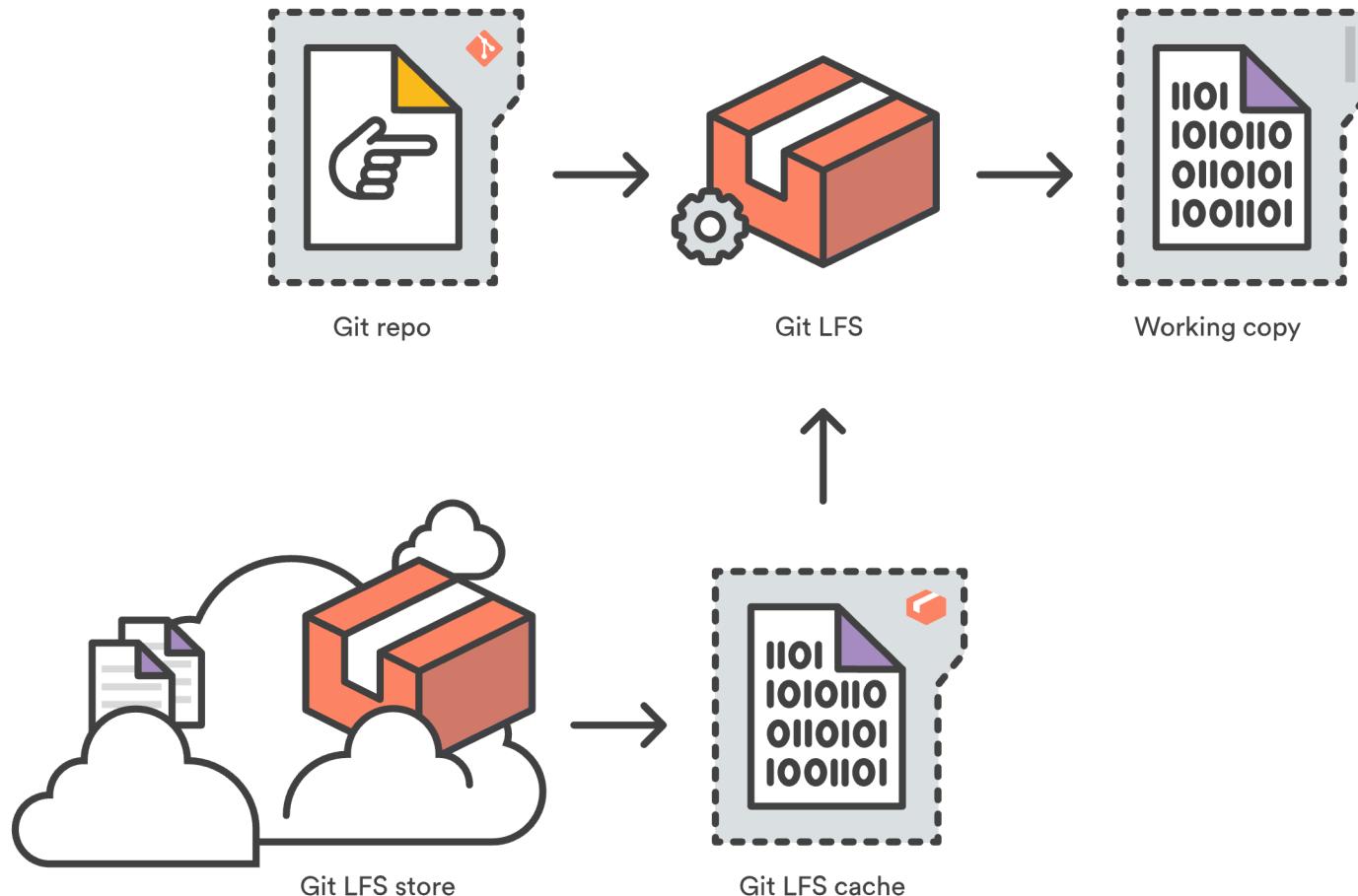
Git LFS

Step 2 : *git push*



Git LFS

Step 3 : git checkout



Mise en pratique

Exercice

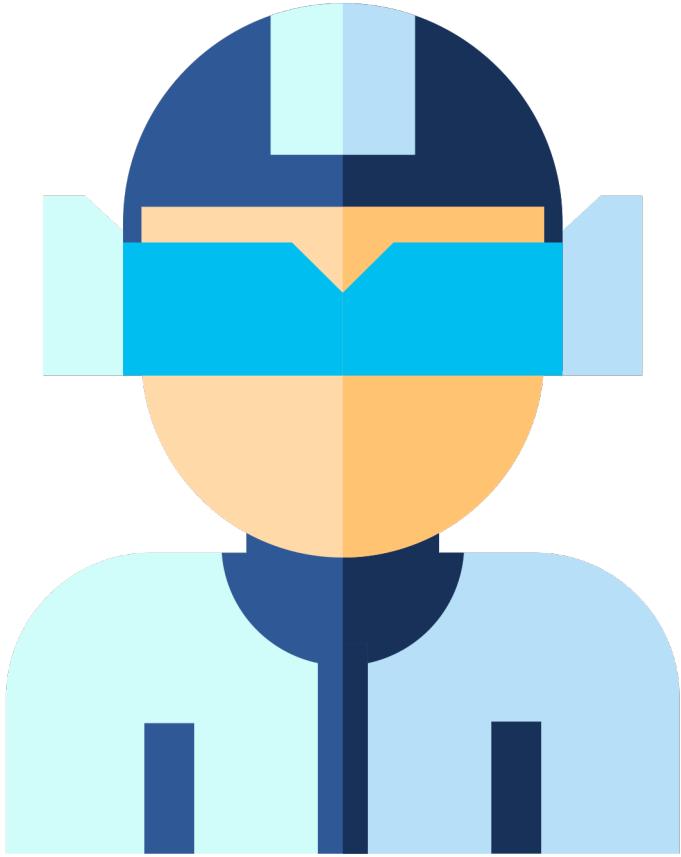
1. Prenez un de vos git, initialisez git-lfs
2. Essayez d'ajouter un gros jpg, poids > 2mo
3. Traquez les « *.jpg » (Repository → git LFS → track/untrack)
4. Faites votre commit + push
5. Confirmez sur GitHub la présence du badge sur le fichier

945 KB |  Stored with Git LFS

6. Téléchargez le zip du repo depuis github
7. Renommez le jpg en txt et vérifiez le poids et le contenu

```
version https://git-lfs.github.com/spec/v1
oid sha256:a43b00361a219e08250bd4f934e993b6153e2bda229414ed468f66f0c784daa3
size 967864
```

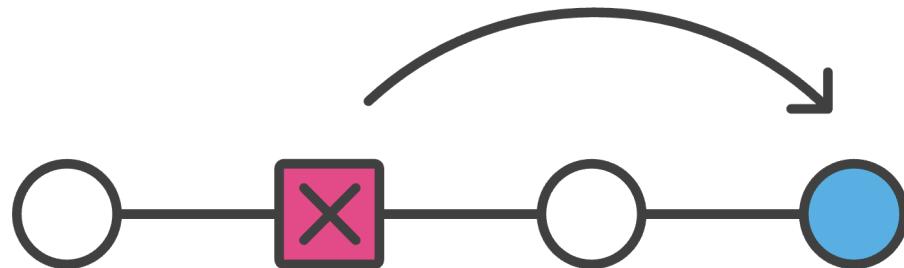
Ex de gros jpg : <https://www.photo-paysage.com/albums/userpics/10001/lac-du-vieux-emosson.jpg>



Pour aller plus
loin

Git revert

- Permet de défaire ce qui a été fait dans une liste de *commit* (pas que le précédent)
- Crée un nouveau *commit*
- Peut demander un *merge*

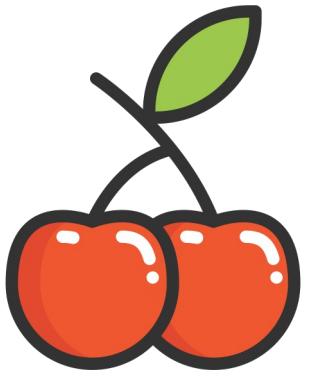


<https://git-scm.com/docs/git-revert>

<https://www.atlassian.com/git/tutorials/undoing-changes/git-revert>

Schema by Atlassian is licensed under CC BY 2.5

Le Cherry Pick

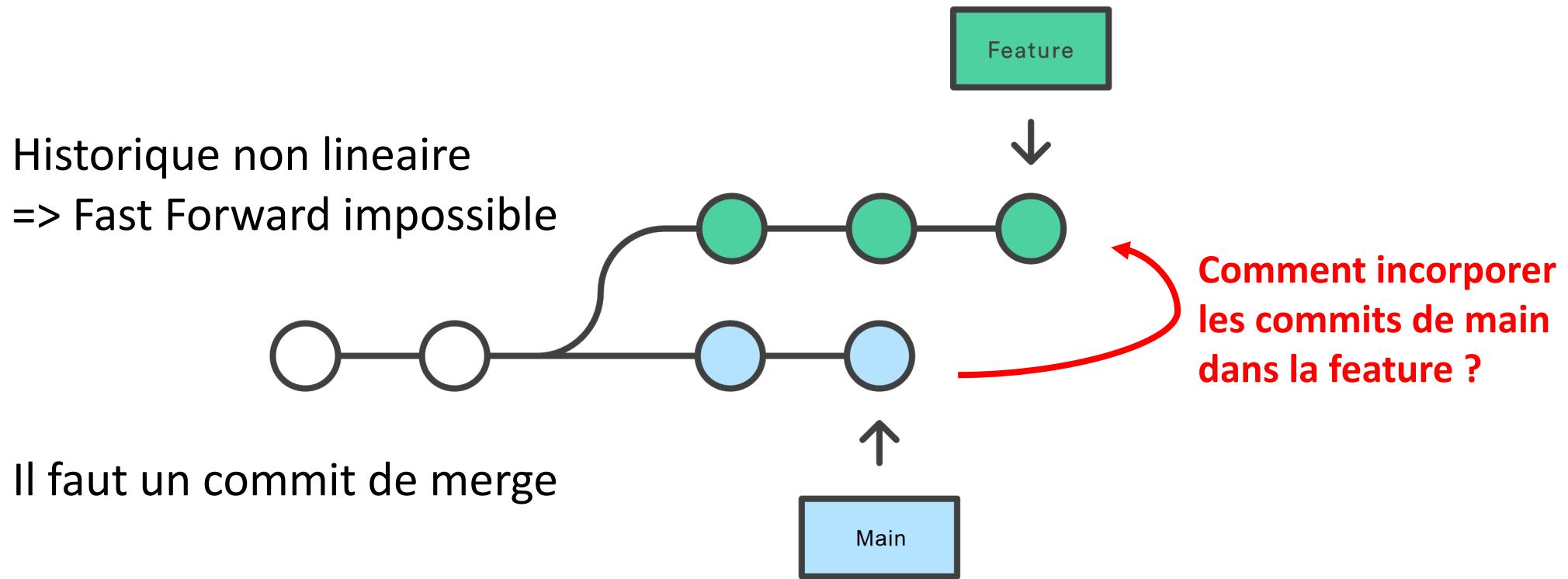


- Permet de sélectionner un commit et de l'appliquer à sa branche actuelle
- Utilité
 - Injecter un fix urgent sans attendre
 - Récupérer du code dans une branche morte/bloquée

⚠️ duplique les commits ⚠️

⚠️ très souvent on préfère le merge ⚠️

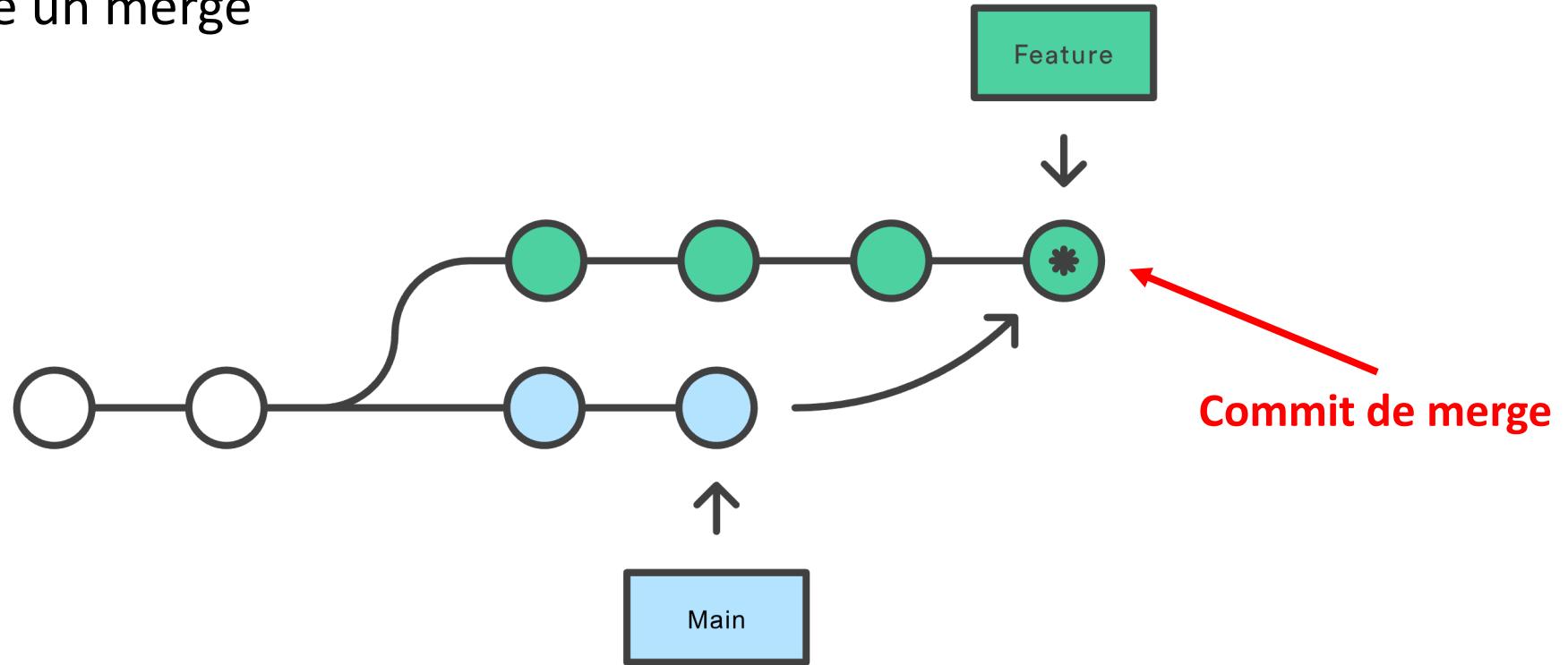
Rebase instead of merge



Schema by [Atlassian](#) is licensed under CC BY 2.5

Rebase instead of merge

Option 1 : faire un merge

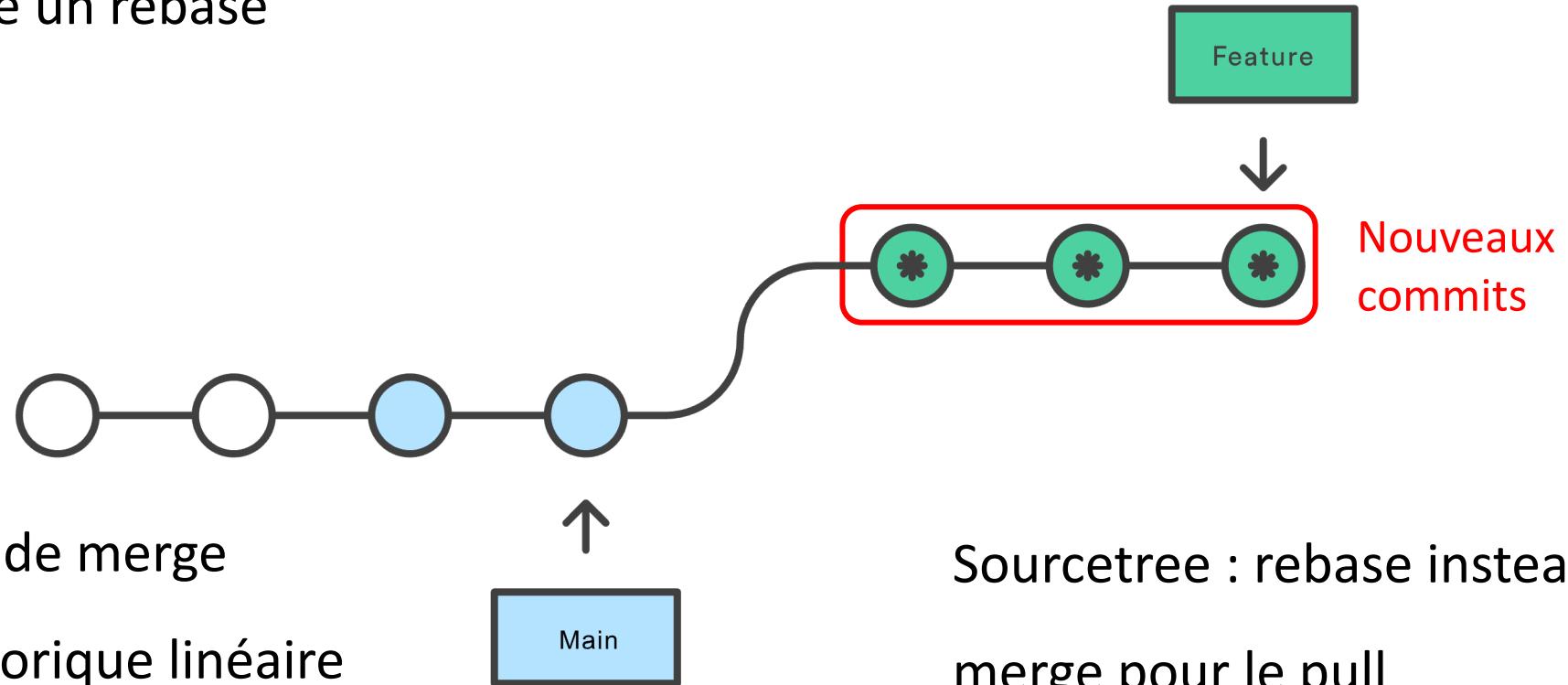


Schema by [Atlassian](#) is licensed under CC BY 2.5

Rebase instead of merge

⚠ danger si mal maîtrisé ⚡

Option 2 : faire un rebase



Src : <https://www.atlassian.com/git/tutorials/merging-vs-rebasing>

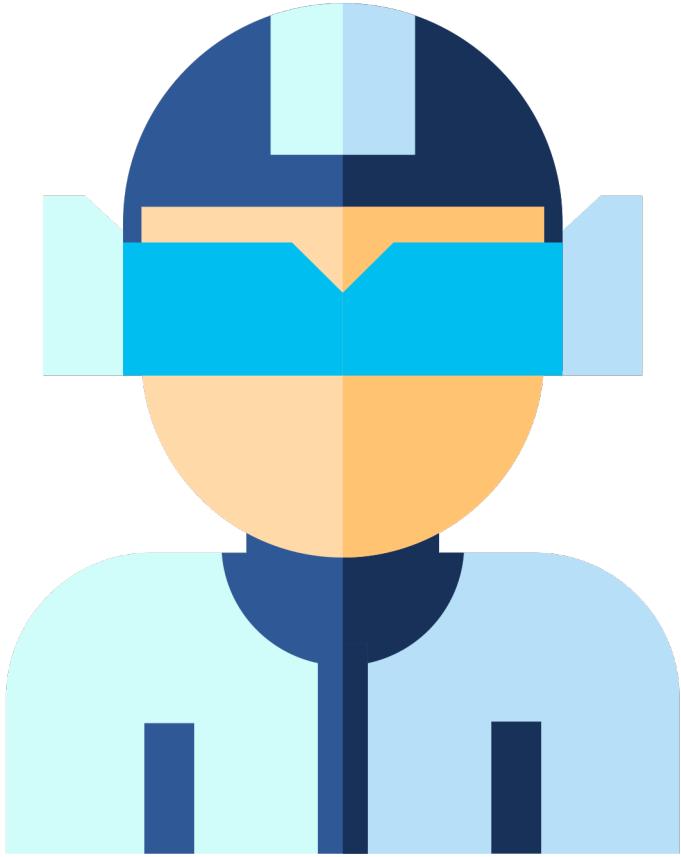
Schema by Atlassian is licensed under CC BY 2.5

Mise en pratique

Exercice

Refaire l'exercice avec les conflits lors du pull mais dans la fenêtre de pull cochez la case « rebase instead of merge »





Pour aller plus
loin

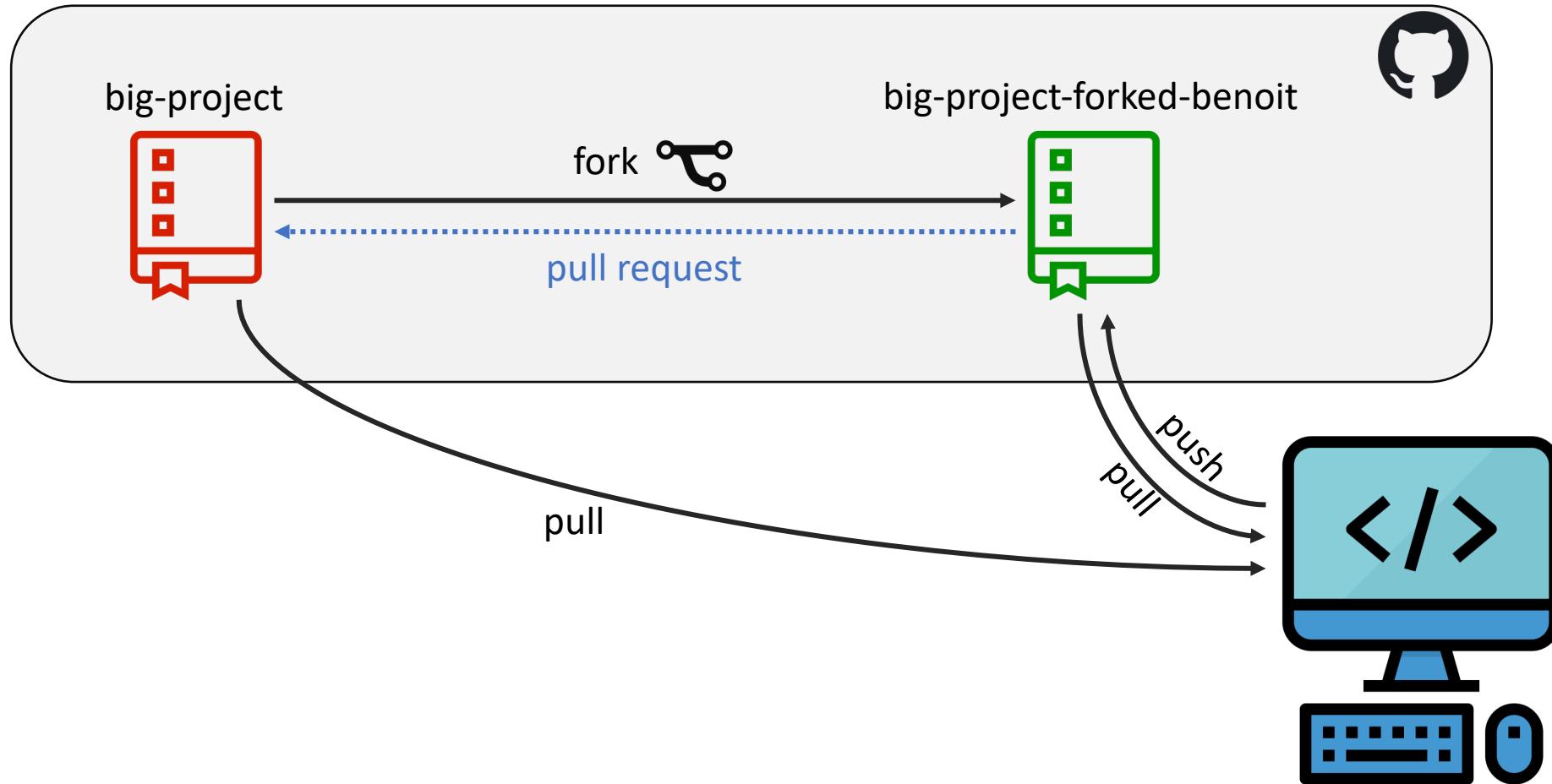
Semantic Versioning

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, and
3. PATCH version when you make backwards compatible bug fixes.

<https://semver.org>

Travailler avec plusieurs remotes





That's all Folks!