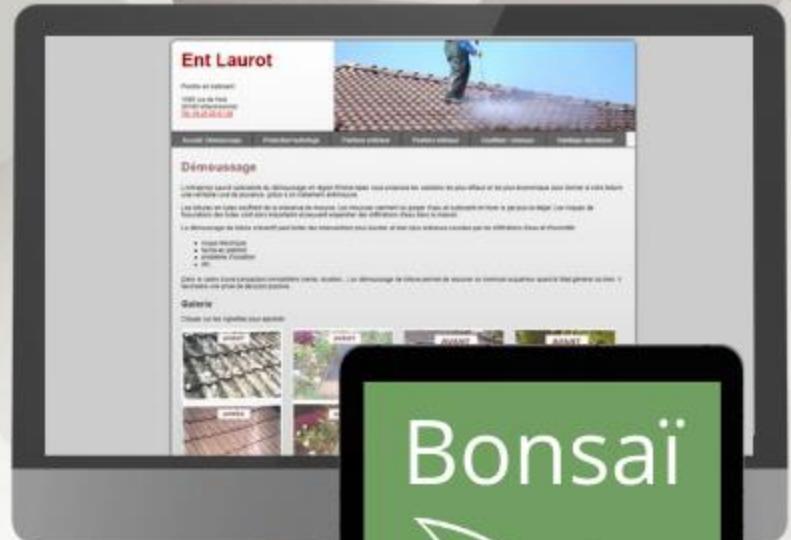


Le versionning avec GIT





pictionAi.fr



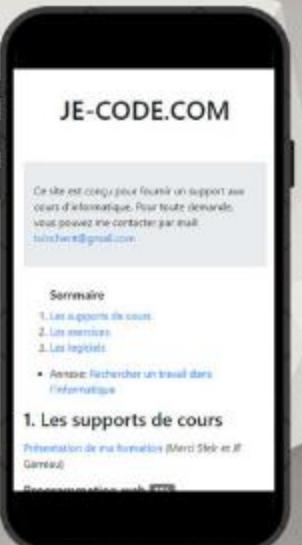
Le pictionary
version IA

Tente de découvrir les mots derrière
l'image issue d'IA à générer.

Qui connait cette image ?



je-code.com



LuxAeterna.fr



Thibault Vincent

Formateur permanent EPSI

thibault.vincent@campus-cd.com

Toujours à votre disposition pour des compléments de cours, suivi de projet, demandes diverses.

- ❖ Formateur en conception d'applications depuis 2015 (écoles d'ingénieur, universités, instituts)
- ❖ Ingénieur développement depuis 2010 (Sopra, Toyota, Eurotunnel etc.).

Illustration : Exemples de sites créés...
... et toujours en fonctionnement !



Programme

- ◊ Les bases de GIT
 - ◊ Utilité de GIT
 - ◊ La ligne de commande
 - ◊ Installation
 - ◊ Les principales commandes (init, commit, push etc.)
 - ◊ Les serveurs GIT : Github, Gitlab, Bitbucket
 - ◊ Le travail en équipe
 - ◊ Le cycle de vie d'un projet
- ◊ GIT avancé
 - ◊ Les bonnes pratiques (régularité des commits, pull)
 - ◊ Le fichier readme avec Markdown
 - ◊ Le .gitignore
 - ◊ Github avancé : github.io, clone, pull request, fork, issues
 - ◊ Le merge conflicts en pratique dans VSCode
 - ◊ Différence head, origin et main
 - ◊ Les commandes avancées : cherry-pick, reset, revert, rebase, blame etc.
 - ◊ Workflow
 - ◊ Github actions, secrets..
- ◊ Bonus : un repo GITHUB style

Les bases de GIT

Utilité, installation,
principales commandes..



Contexte historique

- ❖ Problèmes lors du travail à plusieurs sur un projet
- ❖ Solutions basiques : USB, dossier partagé, SVN
- ❖ Et la solution avec GIT de Linus Torvald



Utilité de GIT

- ❖ Gère l'historique de toutes les modifications d'un projet
- ❖ Facilite les retours en arrière
- ❖ Assure la détection de modifications effectuées par plusieurs personnes sur une même portion de code
- ❖ Décentralisé pour assurer la sauvegarde de données



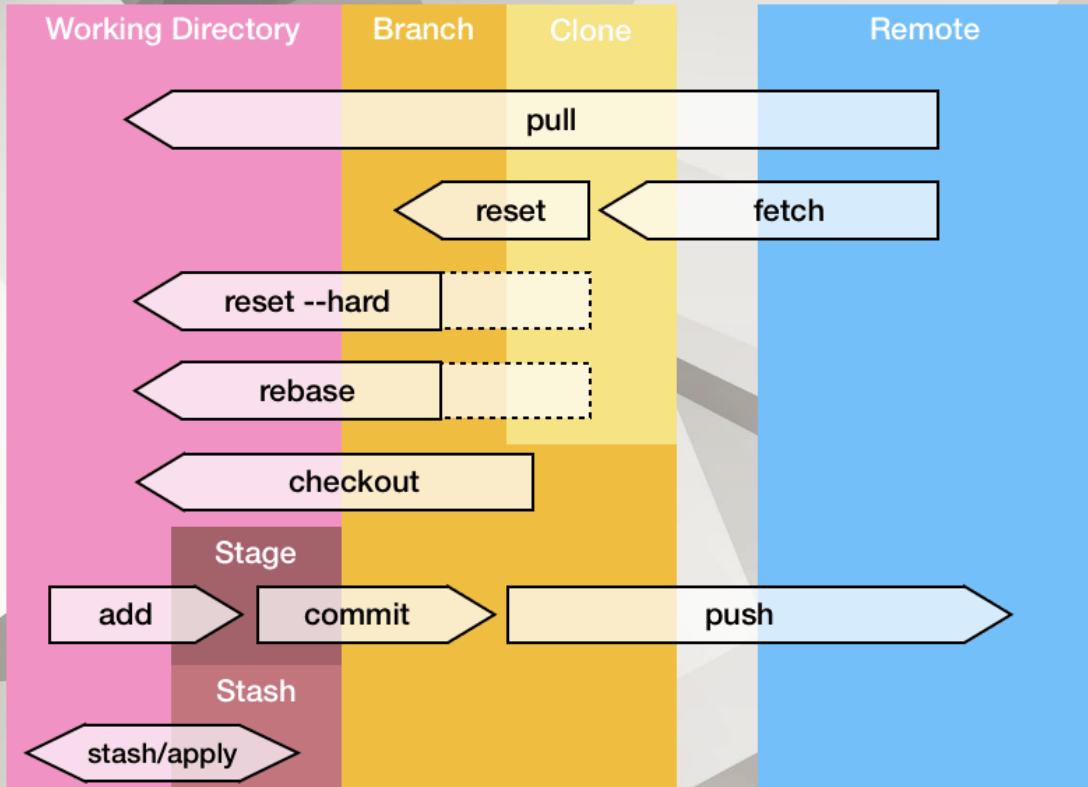
La ligne de commande

- ❖ Permet de faire tout ce qu'on peut faire sur ordinateur, sans interface graphique
- ❖ Sous mac
 - ❖ Application « Terminal »
- ❖ Sous windows
 - ❖ Application « Invite de commande »
- ❖ Les commandes de base:
 - ❖ cd: « change directory »
 - ❖ ls sous mac, dir sous windows
 - ❖ pwd: present working directory
 - ❖ autres commandes de bases: mkdir, mv, rm, chmod, apt-get
- ❖ Astuces:
 - ❖ Utiliser la tabulation pour éviter de taper le nom complet des dossiers et fichiers
 - ❖ Utiliser les flèches haut / bas pour dupliquer une commande faite auparavant



Installation

- ❖ Sous mac : <http://mac.github.com>
- ❖ Sous windows : <http://git-scm.com/download/win>
- ❖ Git s'utilise avec des commandes qui commencent par « git ».
Exemple : git init



Les commandes principales

- ◊ Et aussi :
 - ◊ Init : initialisation d'un repo git
 - ◊ Merge : fusion de travaux
 - ◊ Et d'autres que l'on verra plus tard



Serveur GIT : Github

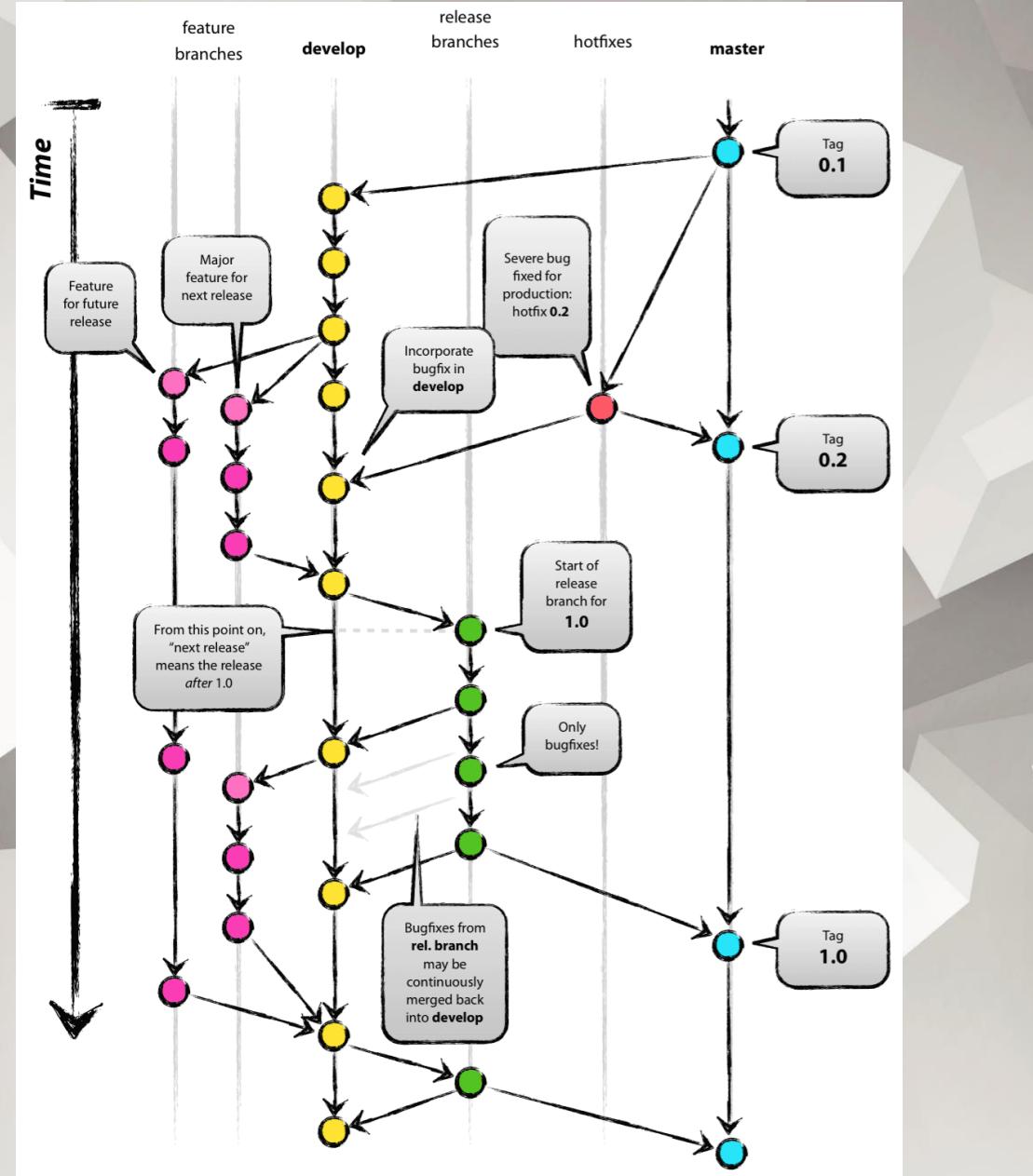
- ❖ Créez votre espace github
 - ❖ <https://github.com/>
 - ❖ New repository puis suivre les instructions
- ❖ Autres espaces : gitlab, bitbucket
- ❖ Attention :
 - ❖ si vous utilisez un ordinateur partagé, il faudra peut-être modifier les paramètres de compte avec git config user.name et user.email
 - ❖ le push ne fonctionne pas si vous vous êtes placé dans un répertoire système de Windows



Le travail en équipe

- ❖ Travailler le projet sur un autre ordinateur
 - ❖ Clone du projet : `git clone`

- ❖ Résoudre les conflits
 - ❖ Vérifier la différence avec le remote : `git status`
 - ❖ Vérifier les différences avec le dernier commit : `git diff`
 - ❖ Fusionner les différences : `git merge`



Le cycle de vie d'un projet GIT

- ❖ La notion de branche
- ❖ Les versions et tag



À vous !

❖ <https://gitexercises.fracz.com/>

GIT avancé

Bonnes pratiques, merge
conflicts, commandes avancées



Quelques bonnes pratiques

- ❖ Faire des commits régulièrement (commit atomic) :
 - ❖ Pas uniquement à chaque fonctionnalité* mais à chaque nouvelle étape fonctionnelle. Exemple : *function* terminée, bug mineur corrigé.
 - ❖ Il ne doit pas se passer plus d'une heure sans commit. 1 commit toutes les 15 minutes est une bonne moyenne.
- ❖ Faire des pull avant de démarrer une nouvelle tâche (rapatrier la dernière version du projet) :
 - ❖ Au début de chaque journée pour récupérer le travail de la veille au soir après son départ.
 - ❖ Après chaque push pour resynchroniser son travail.

* une fonctionnalité de moyenne à grande importance aura en général une branche dédiée.

MARKDOWN

CHEAT SHEET AND NOTEBOOK

Headings

- # H1
- ## H2
- ### H3
- #### H4
- ##### H5
- ##### H6

Emphasis

- *Italic* or Italic
- **Bold** or Bold
- ***Bold and Italic***

Unordered list

- First item
- Second item
- First nested item
- Second nested item
- Third item

Ordered list

1. First item
2. Second item
 - 1. First nested item
 - 2. Second nested item
3. Third item

Links

A link to [Infinite Boop](http://infiniteboop.com "Infinite Boop")

Tables

Column 1	Column 2	Column 3
Row 1, Column 1	Row 1, Column 2	Row 1, Column 3
Row 2, Column 1	Row 2, Column 2	Row 2, Column 3

INFINITE BOOP

Task list

- [x] Task 1
- [] Task 2
- [] Task 3

Images

![Alt Text](Image URL)

Blockquotes

> Lorem Ipsum

Inline code

'Inline Code'

Code block

```
```python
def function():
 print("Hello, world!")
````
```

Horizontal Rule

Strikethrough

~~Strikethrough~~

Le fichier readme avec Markdown

- ❖ Le fichier doit se trouver à la racine du repo et porter le nom « `readme.md` ».
- ❖ Il sera dès lors automatiquement mis en avant sur votre repo github.
- ❖ Le markdown est un fichier texte avec des possibilités de mise en forme très simples. En savoir plus : <https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet>

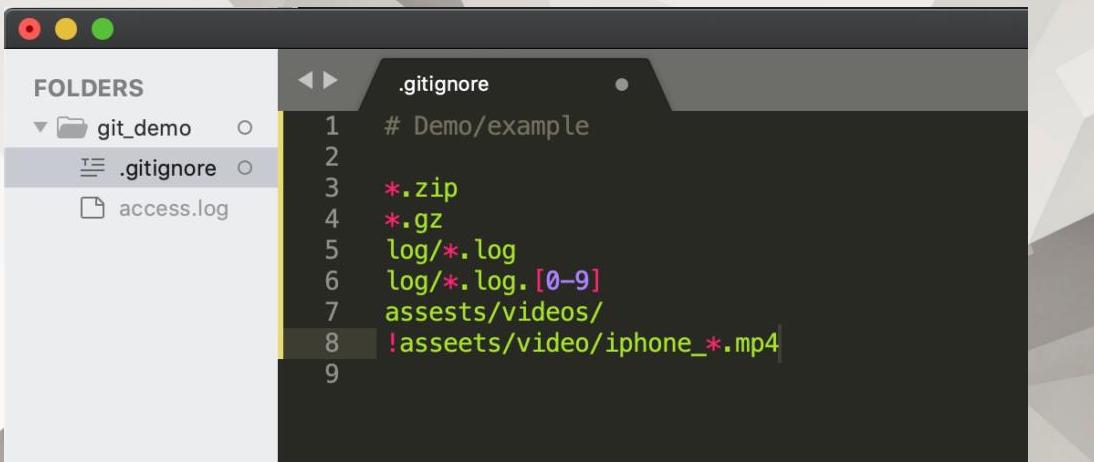


Github avancé

- ❖ **github.io**
 - ❖ Si vous créez un repo ayant pour nom "votrecompte.github.github.io" celui-ci sera automatiquement accessible à l'adresse <https://votrecompte.github.github.io>
 - ❖ Attention, seules les technos front seront actives (HTML, CSS, JS OK), (PHP etc NOK)
- ❖ **Clone**
 - ❖ Commande permettant de récupérer une copie d'un projet sur votre machine
- ❖ **Pull request**
 - ❖ Pull (transfert) d'une modification sur un projet dont vous n'êtes pas propriétaire.
- ❖ **Fork**
 - ❖ Repartir d'un projet existant pour servir de base à un nouveau projet avec le même objectif mais avec une approche différente.
 - ❖ Sert aussi à importer pour modifier un projet qui ne nous appartient pas (en faisant ensuite une pull request).
- ❖ **Issues**
 - ❖ Utilisé pour lister et assigner les todos, bugs en cours, demande d'amélioration etc.

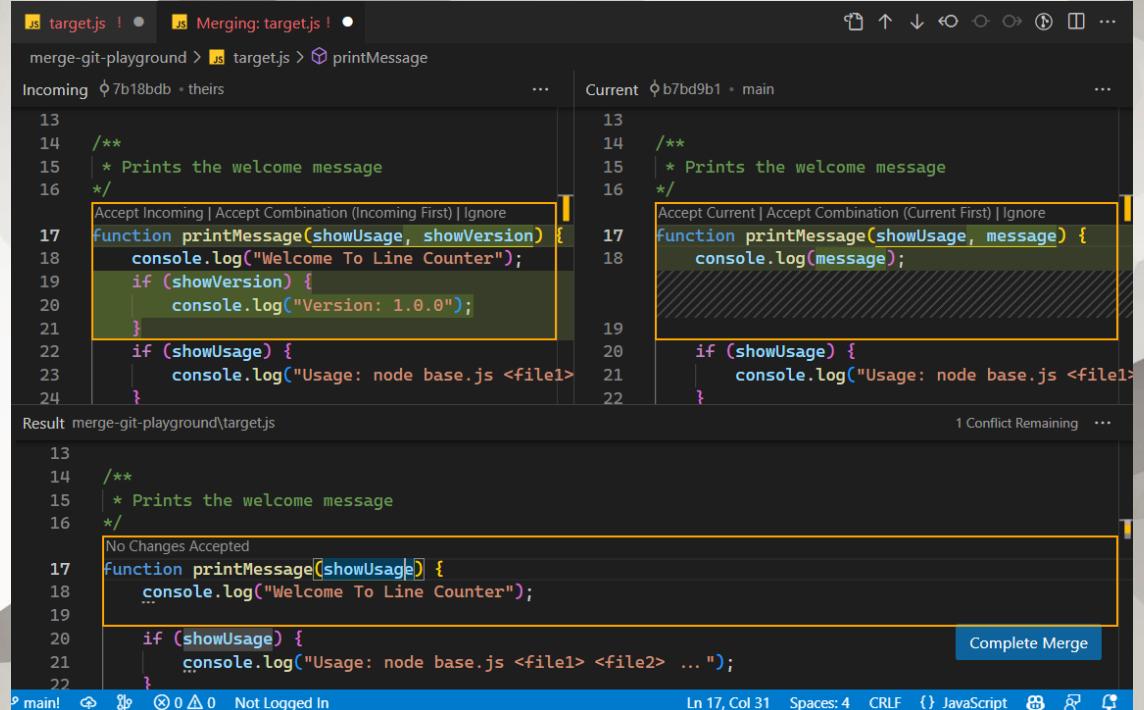
Le fichier .gitignore

- ❖ Fichier de configuration utilisé pour repertorier les dossiers et fichiers qui seront uniquement sur le working directory, et donc exclu du remote



```
git_demo .gitignore access.log
1 # Demo/example
2
3 *.zip
4 *.gz
5 log/*.log
6 log/*.log.[0-9]
7 assets/videos/
8 !assets/video/iphone_*.mp4
9
```

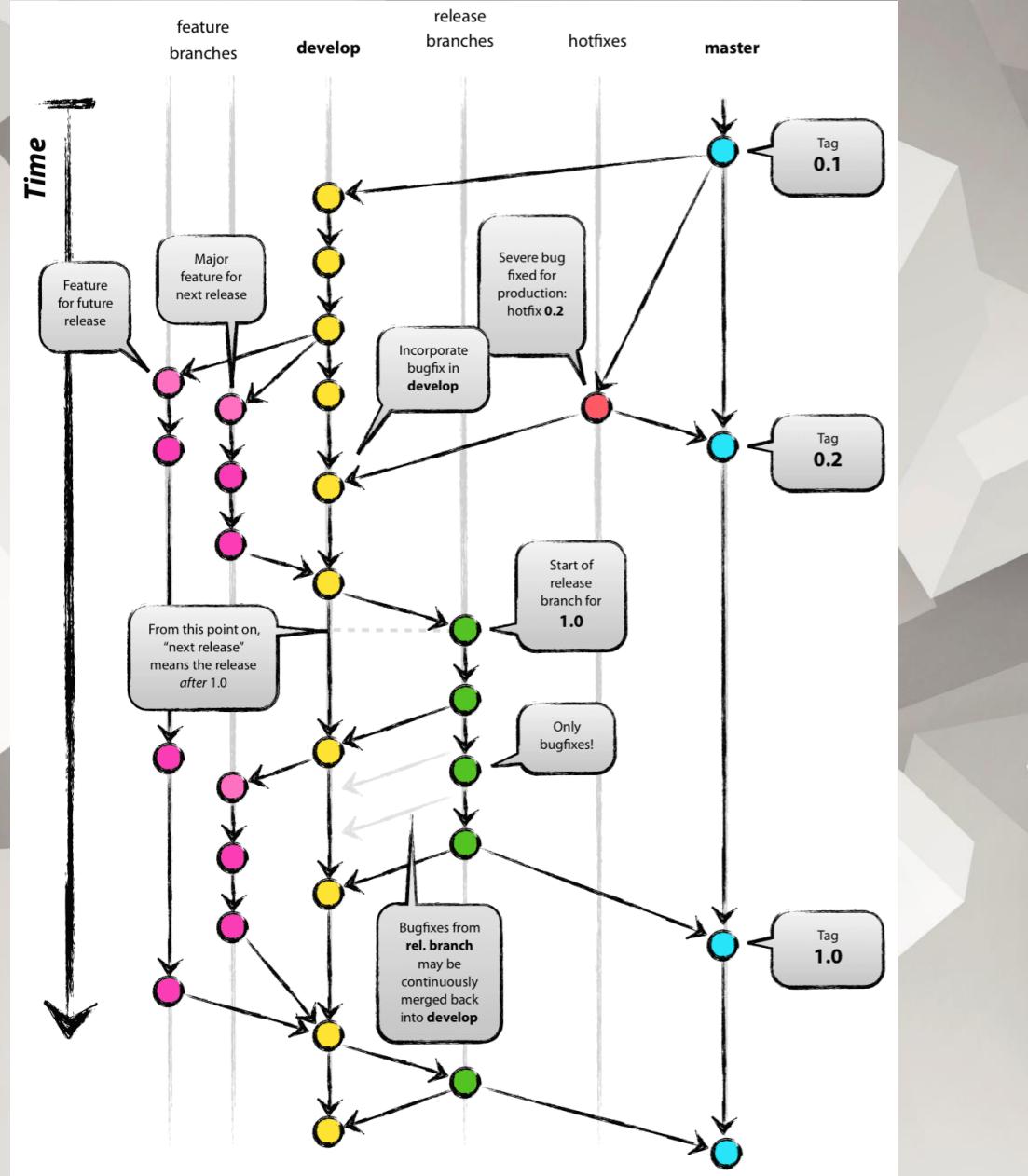
La résolution de conflits



The screenshot shows a code editor with two panes. The left pane is labeled 'Incoming' and the right pane is labeled 'Current'. Both panes show the same code for a `printMessage` function. A conflict is indicated by a yellow box around the line `if (showUsage) {`. The 'Incoming' pane has a tooltip 'Accept Incoming | Accept Combination (Incoming First) | Ignore'. The 'Current' pane has a tooltip 'Accept Current | Accept Combination (Current First) | Ignore'. The bottom pane shows the merged result with the line `if (showUsage) {` removed. A 'Complete Merge' button is visible at the bottom right of the merged pane.

```
target.js ! Merging: target.js !
merge-git-playground > target.js > printMessage
Incoming ⚡ 7b18bdb · theirs
13
14  /**
15   * Prints the welcome message
16  */
17  Accept Incoming | Accept Combination (Incoming First) | Ignore
function printMessage(showUsage, showVersion) {
18    console.log("Welcome To Line Counter");
19    if (showVersion) {
20      console.log("Version: 1.0.0");
21    }
22    if (showUsage) {
23      console.log("Usage: node base.js <file1>");
24    }
Result merge-git-playground\target.js
13
14  /**
15   * Prints the welcome message
16  */
17  No Changes Accepted
function printMessage(showUsage) {
18    console.log("Welcome To Line Counter");
19
20    if (showUsage) {
21      console.log("Usage: node base.js <file1> <file2> ... ");
22
13
14  /**
15   * Prints the welcome message
16  */
17  Accept Current | Accept Combination (Current First) | Ignore
function printMessage(showUsage, message) {
18    console.log(message);
19
20    if (showUsage) {
21      console.log("Usage: node base.js <file1> <file2> ... ");
22
13
14  /**
15   * Prints the welcome message
16  */
17  No Changes Accepted
function printMessage(showUsage) {
18    console.log("Welcome To Line Counter");
19
20    if (showUsage) {
21      console.log("Usage: node base.js <file1> <file2> ... ");
22
Complete Merge
Ln 17, Col 31  Spaces: 4  CRLF  {}  JavaScript  ⚡  Not Logged In
```

- ❖ Rappel :
- ❖ Status
- ❖ Diff
- ❖ Merge



Différences HEAD, ORIGIN, MAIN, MASTER

◆ HEAD

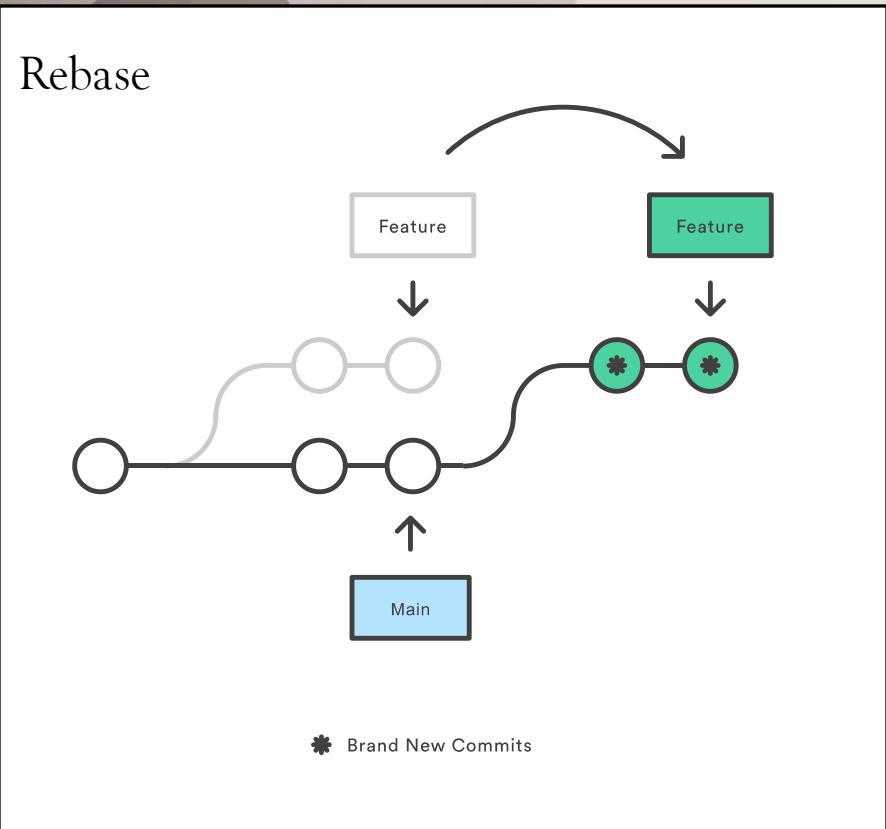
- ◆ Désigne l'endroit où l'on se situe dans l'arborescence de branche.

◆ ORIGIN

- ◆ Désigne le nom par défaut du repo distant (remote)

◆ MAIN

- ◆ Désigne la branche principale, en général utilisée pour les releases principales. Cf schéma
- ◆ Le nom « MASTER » est parfois utilisé à la place de MAIN pour désigner la branche principale.



Les commandes avancées

- ❖ Cherry-pick : importer des éléments d'une autre branche dans la branche courante
- ❖ Reset : annuler les changements pour se positionner à un endroit donné de l'arborescence git
- ❖ Rebase : modifier l'historique des commits pour placer sa base à un autre endroit
- ❖ Revert : inverse les modifications des commit spécifiés, ce qui permet de garder une trace (contrairement à reset)
- ❖ Blame : permet de mettre en évidence qui a développé les lignes de code
- ❖ Stash apply / pop : conserve en mémoire le travail en cours pour y revenir plus tard (avec stash pop – pour supp ou apply – pour garder)



Workflow

- ❖ 2 types de workflow principaux :
 - ❖ Trunk based : tout est sur le main, pas d'autres branches. Rapide mais très exigeant. Demande beaucoup de rigueur comme la nécessité de faire des commit atomic). Peut bloquer la phase de test.
 - ❖ Gitflow : moins exigeant et moins risqué mais plus long à maintenir.
- ❖ Commencer donc par le gitflow..



Github actions, secrets..

- ❖ Github actions (CI/CD)
 - ❖ Fichier .yml qui doivent se trouver dans le dossier .github/workflows
 - ❖ S'execute à chaque nouveau push
 - ❖ Utile pour mettre en place un pipeline automatisé. Exemple : lorsque je push mes modifications, le projet exécute automatiquement les tests et si les tests réussissent, les poussent sur le serveur de recette.
- ❖ Secrets
 - ❖ Dans Settings / Secrets and variables
 - ❖ Pratique pour stocker des variables d'environnements
 - ❖ Qui sont utile sur le remote
 - ❖ Mais qui ne doivent pas être visible du public
 - ❖ Exemple : le mot de passe du serveur de FTP utilisé pour la mise en ligne



YoanBor

yoanbor

Unfollow

3 followers · 9 following

Achievements



Beta Send feedback

Block or Report

yoanbor / README.md

Hi there !

About Me:

Je suis un jeune passionné d'informatique actuellement en formation chez Simplon Valenciennes pour devenir Concepteur Développeur d'Applications (diplôme de niveau 6). Dans le cadre de cette formation, je suis à la recherche d'un stage de 3 mois (non rémunéré) du 13 mai 2024 au 06 août 2024.

[LinkedIn](#)

Tech Stack:

Front:

HTML5, CSS3, SASS, JAVASCRIPT, TYPESCRIPT, REACT, ANGULAR

Back:

JAVA, PHP

Database

WINDOWS TERMINAL, MONGODB, MYSQL, POSTGRES, MARIADB, MICROSOFT SQL SERVER

CI/CD

DOCKER, JIRA, TRELLO, NOTION

Bonus

- ❖ Un repo stylé avec <https://gprm.itsvg.in/>
- ❖ Génère un fichier readme.md à déposer dans le repo « nomDeProfilGithub »



À vous !

❖ https://learngitbranching.js.org/?locale=fr_FR



MAJ 2026

- ❖ Éviter checkout qui est déprécié et lui préférer switch (changement et création de branche) ou restore
- ❖ Convention de nommage des branches largement adoptée :
 - main : production
 - develop : intégration
 - feature/xxx
 - fix/xxx
 - Hotfix/xxx
- ❖ Pull requests obligatoires : Jamais de push direct sur main: Revue de code, Tests automatiques, Historique maîtrisé
- ❖ CI systématique : Chaque push déclenche : Tests, Lint, Build
- ❖ Versionner les livraisons :
 - git tag v1.2.0
 - git push --tags