

Estimation des tâches

Estimation 2

Créer un petit maison

X Point(s)



0	$\frac{1}{2}$	1	2
3	5	8	13
20	40	100	∞
?	☕		

Estimation des tâches

Estimation 3

Créer un programmable robot

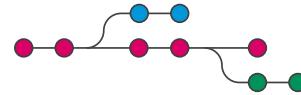
X Point(s)



0	$\frac{1}{2}$	1	2
3	5	8	13
20	40	100	∞
?	☕		



Conception des systèmes Version Control



Filière Systèmes industriels

Silvan Zahno silvan.zahno@hevs.ch



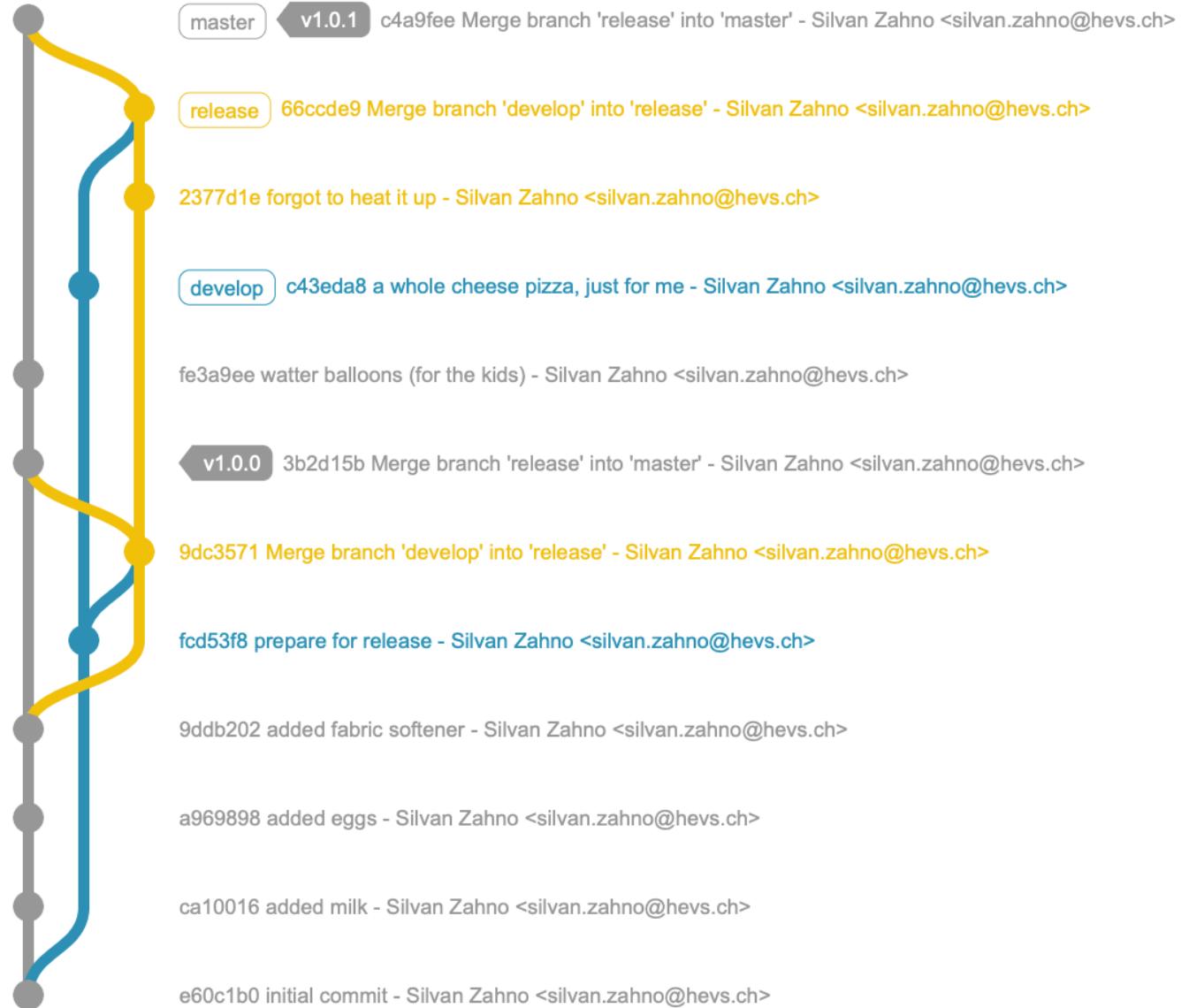
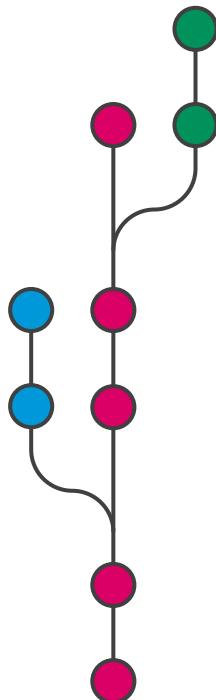


Votre système actuel

```
•
├── fichier important copy.txt
├── fichier important v1.txt
├── fichier important v2.1 backup.txt
├── fichier important v2.1 backup.txt.old
├── fichier important v2.1.txt
├── fichier important v2.txt
└── fichier important.txt
```

1 répertoire, 7 fichiers

Version control



Outils possibles

Git (git)



Subversion (svn)



Mercurial (hg)

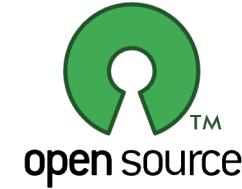
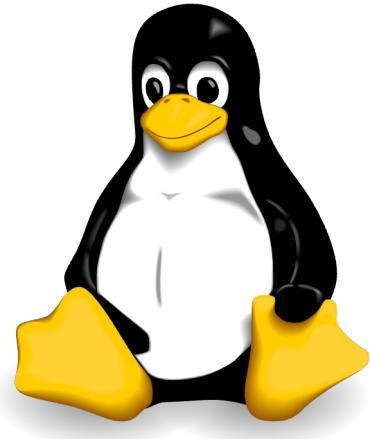


mercurial

SyD Version Control

Linux Torvalds

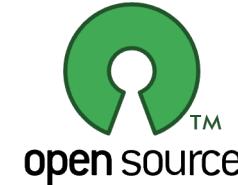
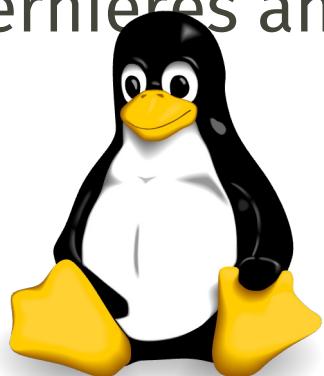
Linux (1991)



Git (2005)

Pourquoi Linux a besoin de git

Linux est devenu le plus grand projet de développement collaboratif de l'histoire de l'informatique au cours des 30 dernières années.



- **600 distributions Linux actives**
- **85% de tous les smartphones**
- **500 superordinateurs de pointe**
- **> 27,8 millions de lignes de code**
- **> 12 000 contributeurs**
- **> 1 million de commits**

<https://truelist.co/blog/linux-statistics/>

Plateformes Git



Gitlab

<https://gitlab.com>

<https://gitlab.hevs.ch>



Github

<https://github.com>



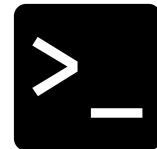
Bitbucket

<https://bitbucket.com>

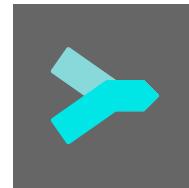


Outils Git

Ligne de commande



Sublime Merge



Git Cola



Git Kraken



Fork



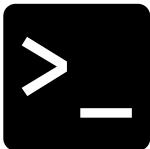
Tower



SmartGit



Git ligne de commande



```
git status  
git diff  
git add <file>  
git diff --staged  
git reset <file>  
git commit -m "<commit message>"  
git fetch <remote>  
git merge <remote> <branch>  
git push <remote> <branch>  
git pull
```

```
Last login: Tue Mar  8 09:26:26 on ttys004
[zas@zac ~] (base)
$ git --help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--super-prefix=<path>] [--config-env=<name>=<envvar>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
clone   Clone a repository into a new directory
init    Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
add     Add file contents to the index
mv      Move or rename a file, a directory, or a symlink
restore Restore working tree files
rm      Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
bisect  Use binary search to find the commit that introduced a bug
diff    Show changes between commits, commit and working tree, etc
grep    Print lines matching a pattern
log     Show commit logs
show   Show various types of objects
status  Show the working tree status

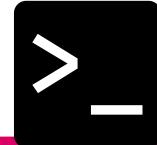
grow, mark and tweak your common history
branch  List, create, or delete branches
commit  Record changes to the repository
merge   Join two or more development histories together
rebase  Reapply commits on top of another base tip
reset   Reset current HEAD to the specified state
switch  Switch branches
tag     Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
fetch   Download objects and refs from another repository
pull    Fetch from and integrate with another repository or a local branch
push    Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.

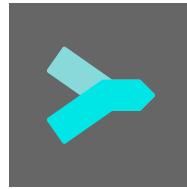
[zas@zac ~] (base)
$
```

git commands



Command	Description	Command	Description
Démarrer une zone de travail		Développez, marquez et modifiez votre histoire commune	
clone	Cloner un dépôt dans un nouveau répertoire	branch	Lister, créer ou supprimer des branches
init	Créer un dépôt Git vide ou réinitialiser un dépôt existant	checkout	Changer de branche ou restaurer les fichiers de l'arborescence de travail
Travailler sur la modification en cours		commit	Enregistrer les modifications apportées à la base de données
add	Ajouter le contenu d'un fichier à l'index	diff	Afficher les changements entre les livraisons, les livraisons et l'arbre de travail, etc.
mv	Déplacer ou renommer un fichier, un répertoire ou un lien symbolique	merge	Joindre deux ou plusieurs historiques de développement
reset	Réinitialiser le HEAD actuel à l'état spécifié	rebase	Réappliquer les commits par-dessus un autre conseil de base
rm	Supprimer des fichiers de l'arbre de travail et de l'index	tag	Créer, lister, supprimer ou vérifier un objet d'étiquette
Examiner l'historique et l'état		Collaborer	
log	Afficher les journaux de livraison	fetch	Télécharger des objets et des références à partir d'un autre dépôt
show	Afficher différents types d'objets	pull	Récupérer et intégrer des objets d'une autre base de données ou d'une branche locale
status	Afficher l'état de l'arbre de travail	push	Mettre à jour les références distantes ainsi que les objets associés

Sublime Merge

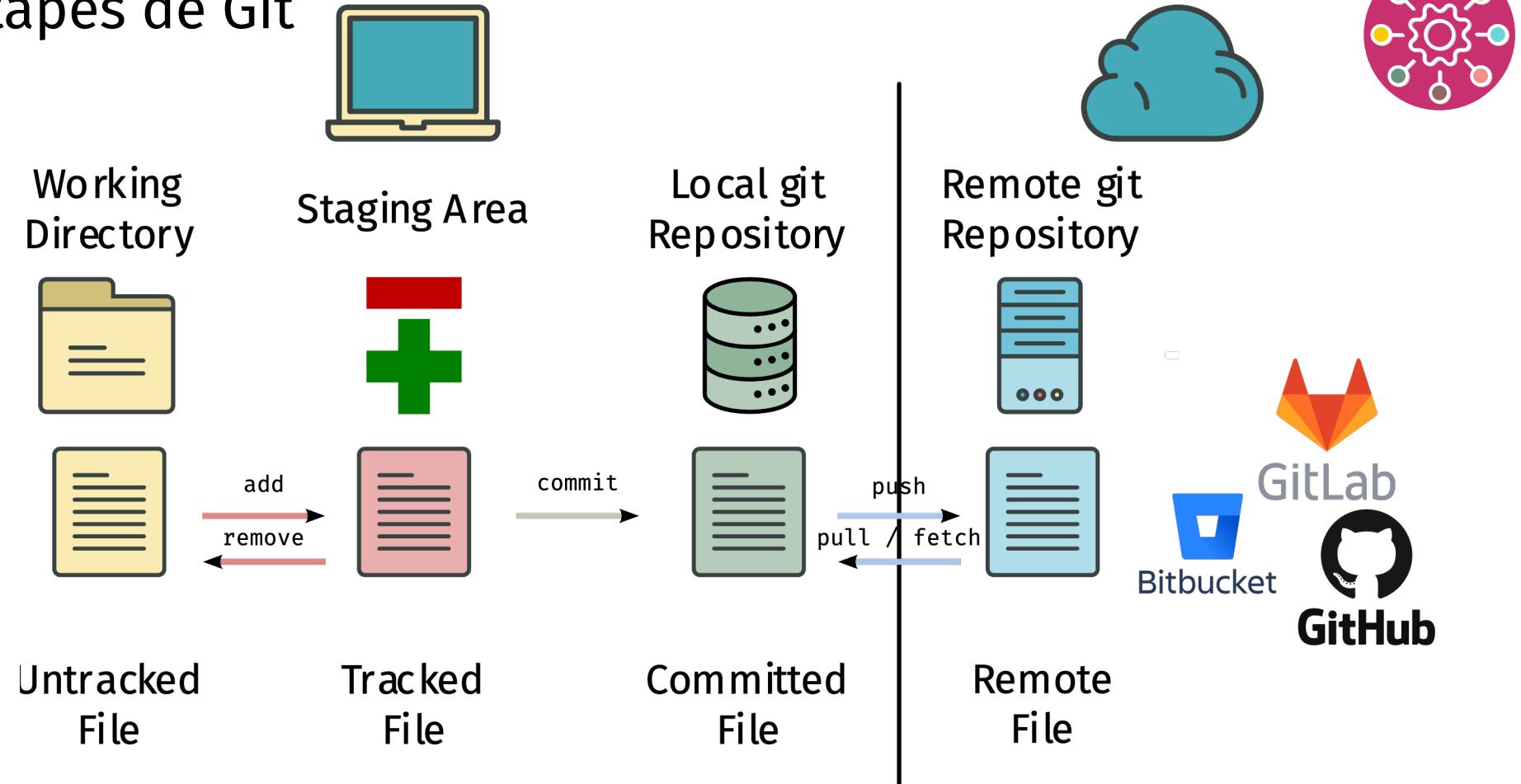


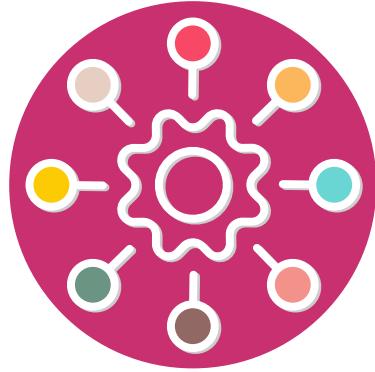
The screenshot shows a GitHub repository interface for a project named 'car-course'. A pull request titled 'Merge remote-tracking branch 'origin/master'' is open. The pull request summary indicates 16 commits, 16 files changed, and 16 additions/10 deletions. The commits are listed in a timeline, with some collapsed. Key commits include:

- Merge remote-tracking branch 'origin/master'
- CHG: updated planning
- ADD: ALU and ImmSrc doc
- ADD: EBS2/EBS3 specs
- FIX: memory stack images
- FIX: errors in immediate and type images
- ADD: files in arc exercises
- ADD: note on Ripes memory management
- CHG: Planning
- FIX: reverse engineering solution
- FIX: ISA syntax errors
- Merge remote-tracking branch 'origin/master'
- FIX: errors in ISA
- ADD: windows Geekbench window
- FIX: add scripts folder
- REM: car-heirv and car-labs doc deployment
- CHG: BEM labo from geekbench 5 to 6
- UPD: all PDFs
- FIX: errors in ARC, ISA, FUN and PER slides

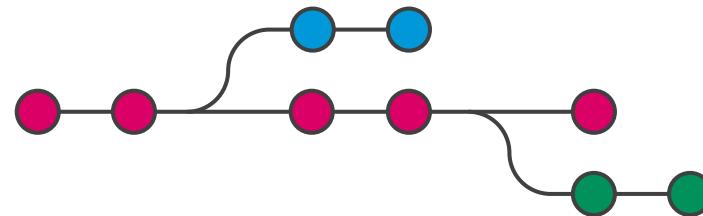
The commit details for 'CHG: BEM labo from geekbench 5 to 6' are shown in more detail, including the commit message, author, date, parents, branches, and stats. The commit message includes a link to a LaTeX document: [labo/latex/02-content/scr/00-solution.tex](#). The commit also contains a note about instruction timing: "Each instruction takes one clock cycle => 19 are executed (addi x5, x0, 0 not executed because of previous beq, and addi x2, x0, 1 not executed because of jal). => 19/664 = 287.9 ns." The commit also includes a LaTeX code snippet for simulation.

Étapes de Git





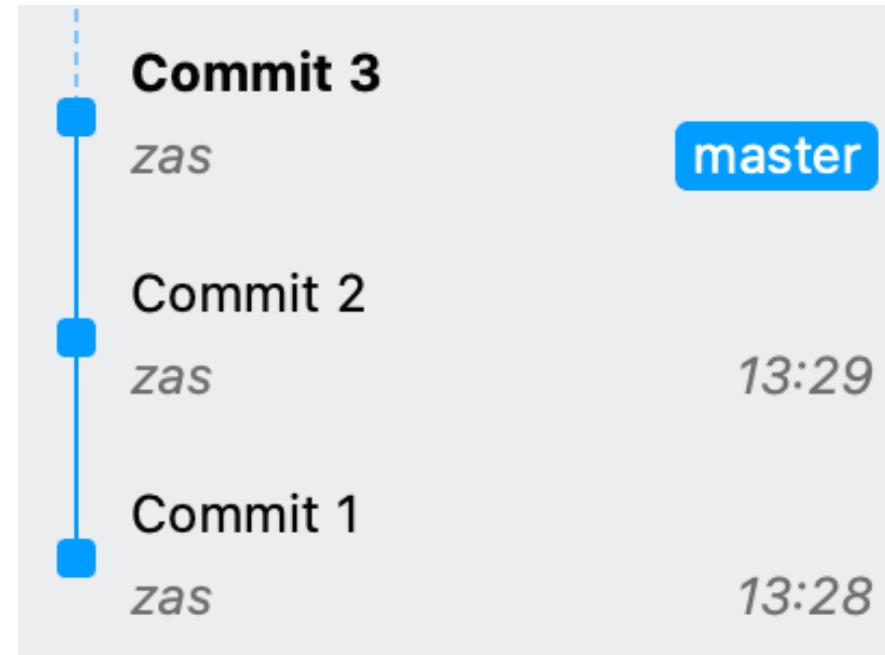
Git Branch et Merge Exemple



Branch et Merge

Initial repo state

Chaque repo a une branche **main** ou une branche **master** comme branche par défaut.



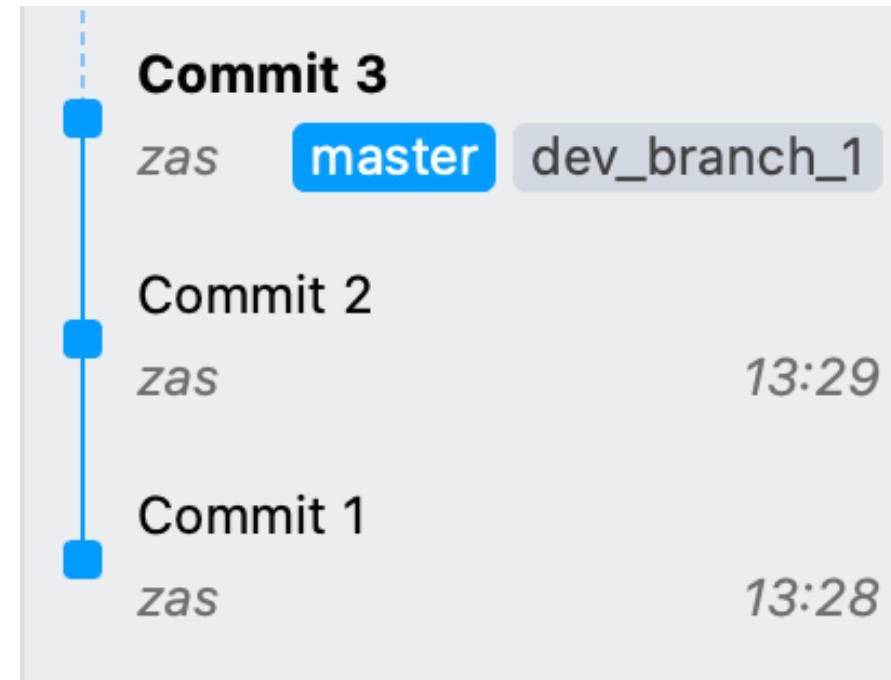
Branch et Merge

Create branch dev_branch_1



Create branch dev_branch_1

```
$ git branch dev_branch_1
```





Branch et Merge

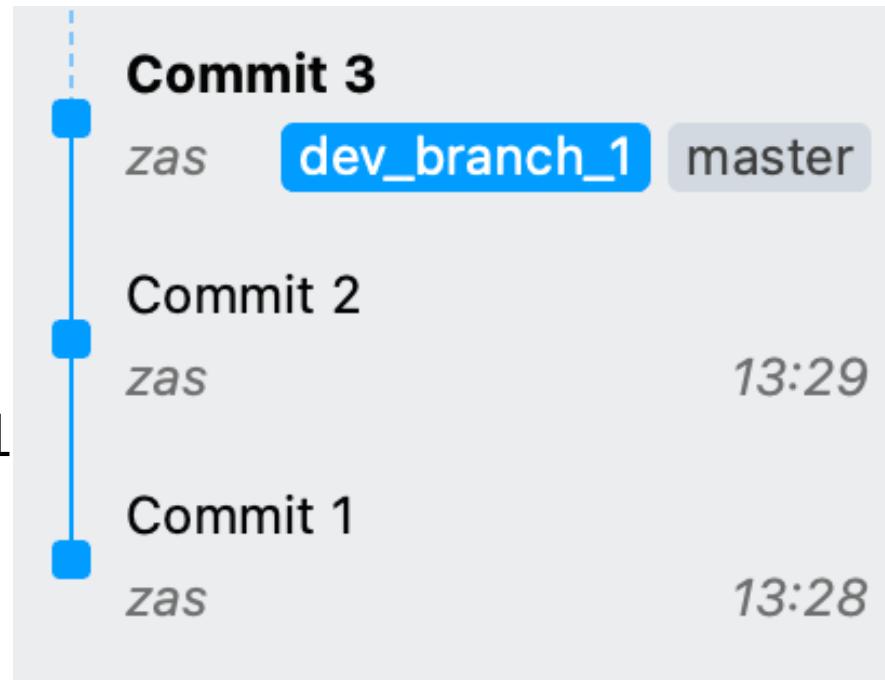
Checkout branch dev_branch_1

Vérifier sur quelle branche nous nous trouvons

```
$ git branch
```

Checkout branch dev_branch_1

```
$ git checkout dev_branch_1
```





Branch et Merge

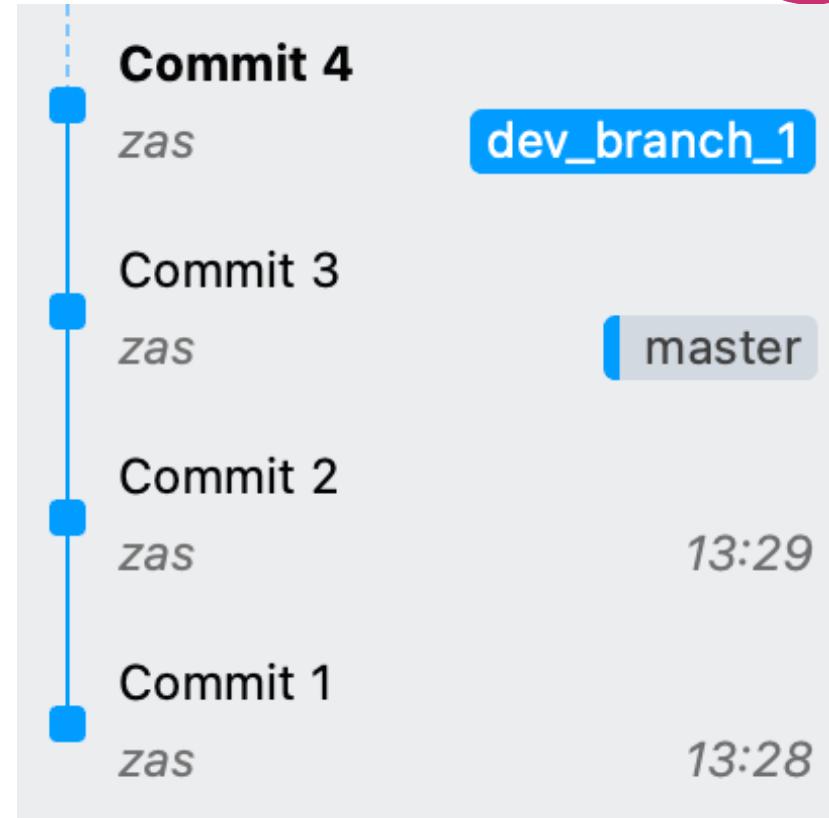
Commit on dev_branch_1

Stage new file

```
$ git add file.md
```

Commit stages files

```
$ git commit -m "Commit 4"  
"
```





Branch et Merge

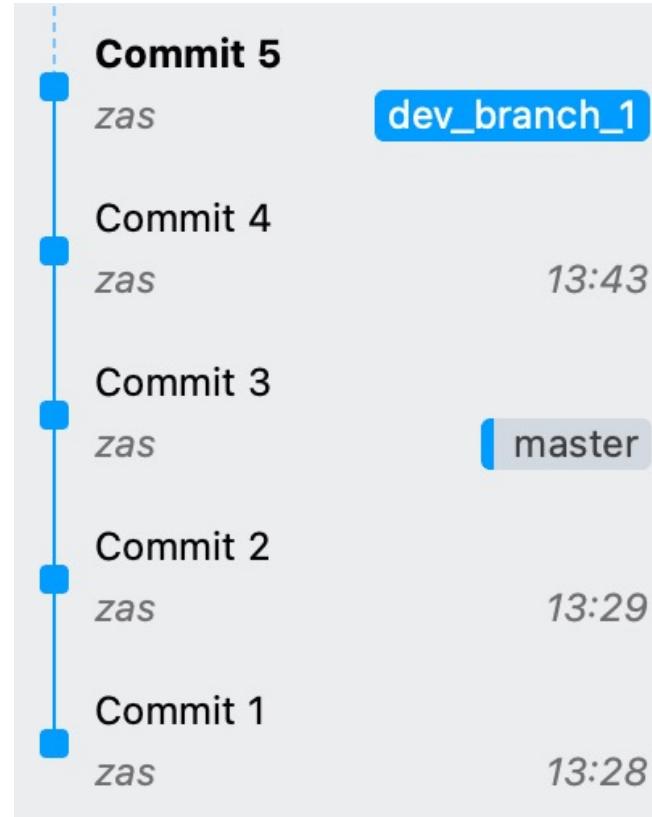
Commit on dev_branch_1

Stage new file

```
$ git add file.md
```

Commit stages files

```
$ git commit -m "Commit 5"
```





Branch et Merge

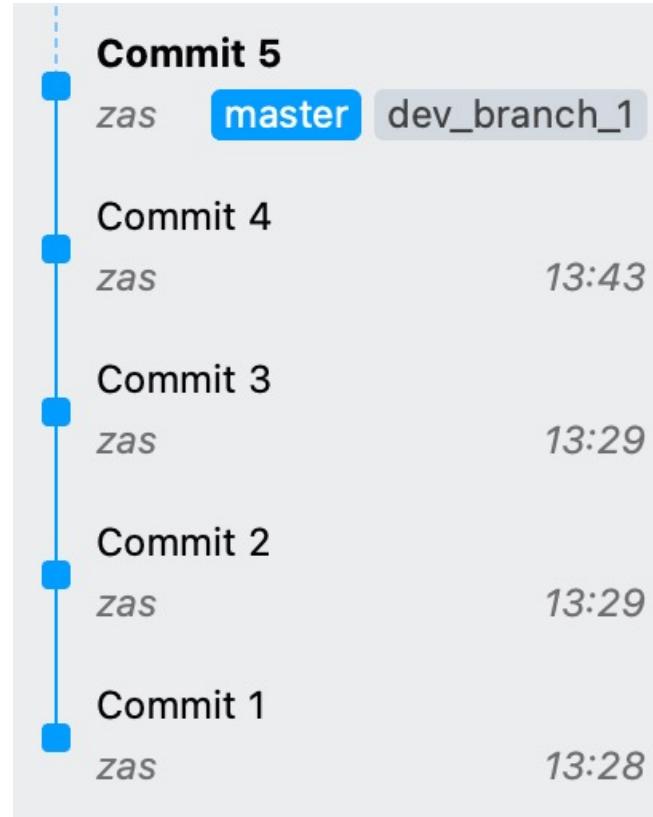
Merge master and dev_branch_1

Checkout master branch

```
$ git checkout master
```

Merge dev_branch_1 into master

```
$ git merge dev_branch_1
```





Branch et Merge

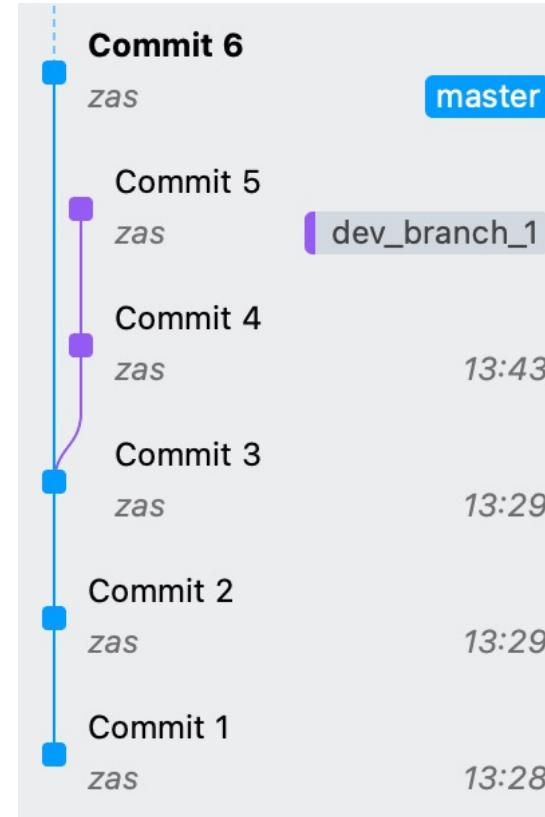
Commit on master branch

Stage new file

```
$ git add file.md
```

Commit stages files

```
$ git commit -m "Commit 6"
```



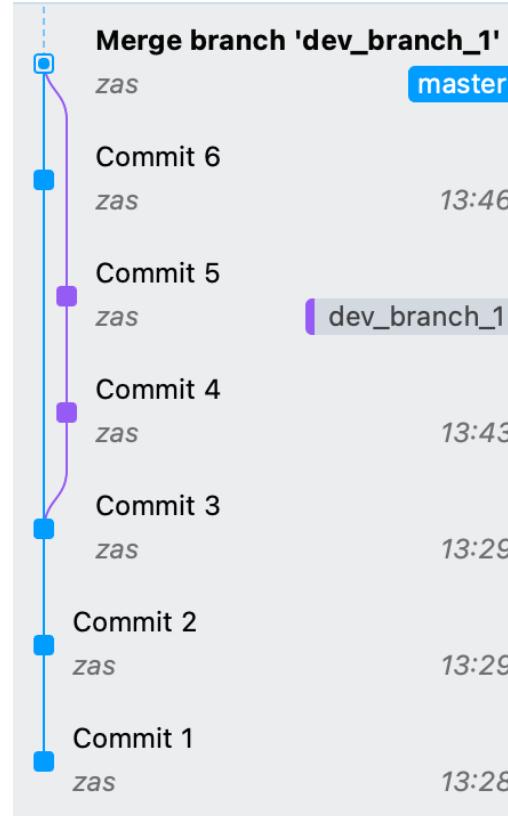
Branch et Merge

Three way merge master and dev_branch_1

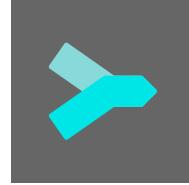


Merge dev_branch_1 into master

```
$ git merge dev_branch_1
```



Demo



■ -/work/repo/edu/car/car-course

LICENSE UPGRADE REQUIRED

did-course car-course car-exams znotes syd-course

Commits Files Summary

BRANCHES (1) master

REMOTES (1) origin

TAGS (0)

STASHES (0)

SUBMODULES (0)

Commit Hash: 702c8f1738addb639884c48b724ab86361d3c
Tree: 702c8f1738addb639884c48b724ab86361d3c
Author: zas <silvan.zahn@hevs.ch>
Date: Thu, 20 Apr 2023 08:12
Parents: 6e927fe, 68810079
Branches: master origin/master
Stats: 16 files changed: 15 +110

Merge remote-tracking branch 'origin/master' (16)

CHG: updated planning (4)

ADD: ALU and ImmSrc doc (4)

ADD: EBS2/EBS3 specs (4)

FIX: memory stack images (5)

FIX: errors in immediate and type images (44)

ADD: files in arc exercises (33)

ADD: note on Ripes memory management (11)

CHG: Planning (4)

FIX: reverse engineering solution (8)

FIX: ISA syntax errors (3)

Merge remote-tracking branch 'origin/master' (16)

CHG: BEM labo from geekbench 5 to 6 (14)

UPD: all PDFs (30)

FIX: errors in ARC, ISA, FUN and PER slides (26)

Thu 08:11

Thu, 13 Apr 15:19

Tue, 11 Apr 15:25

Tue, 4 April 11:00

Tue, 4 April 07:46

Mon, 3 April 13:30

Fri, 31 Mar 15:38

Fri, 31 Mar 08:45

Thu, 30 Mar 17:25

Tue, 28 Mar 07:59

Thu, 28 Mar 07:29

Thu, 16 Mar 10:14

Thu, 16 Mar 09:31

Tue, 14 Mar 14:25

Wed, 8 Mar 19:10

Thu, 7 Mar 09:09

Summary: 00-solution.tex 03-controlunit.tex 04-simulation.tex 05-deployment.tex 05-deployment_ebs3.tex 08-CAR-Labor-SCR-d.pdf 08-CAR-Labor-SCR-f.pdf CAR-Labor-SCR-s.pdf CAR-Labor-SCR-t.pdf

Subsection: Simulation:

```
164 done;
165 beq x2, x2, main      # infinite loop
166 end(minted)
167
168 Each instruction takes one clock cycle => 19 are executed (addi x5, x0, 0 not
169 executed because of previous beq, and addi x2, x0, 1 not executed because of jal).
170 => 19/65M = 287.9 ns.
171 On the EBS2 board @ 60MHz \rightarrow t_{exec} = \frac{nb\_cycles}{f_{sys}} = \frac{19}{60M} = 287.9 ns.
172 On the EBS3 board @ 50MHz \rightarrow t_{exec} = \frac{nb\_cycles}{f_{sys}} = \frac{19}{50M} = 380 ns.
173
174 begin(center)
175   centerLine{\includegraphics[width=0.9\paperwidth]{scr/sol/simulation.pdf}}
176
```

Subsection: Umsetzung:

```
63 Le \textbf{mainDecoder} peut être écrit en VHDL. Pour cela, vous pouvez analyser
64 l'exemple de code \textbf{riscv-mainDecoder-code} ci-dessous et l'adapter en
65 conséquence.
66
67 \begin{center}
68 \begin{tikzpicture}[font=\scriptsize]
69   Schreiben Sie hierzu für beide Subblöcke, \textbf{mainDecoder} sowie \textbf{ALUdecoder}, eine Wahrheitstabelle für alle benötigten Instruktionen.
70 \end{center}
```

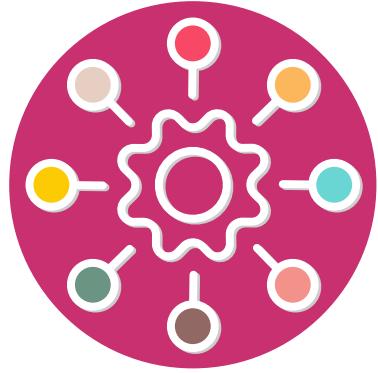
Subsection: Umsetzung:

```
63 Le \textbf{mainDecoder} peut être écrit en VHDL. Pour cela, vous pouvez analyser
64 l'exemple de code \textbf{riscv-mainDecoder-code} ci-dessous et l'adapter en
65 conséquence.
66
67 \begin{center}
68 \begin{tikzpicture}[font=\scriptsize]
69   Schreiben Sie hierzu für beide Subblöcke, \textbf{mainDecoder} sowie \textbf{ALUdecoder}, eine Wahrheitstabelle für alle benötigten Instruktionen.
70 \end{center}
```

fig:riscv-mainDecoder-code:

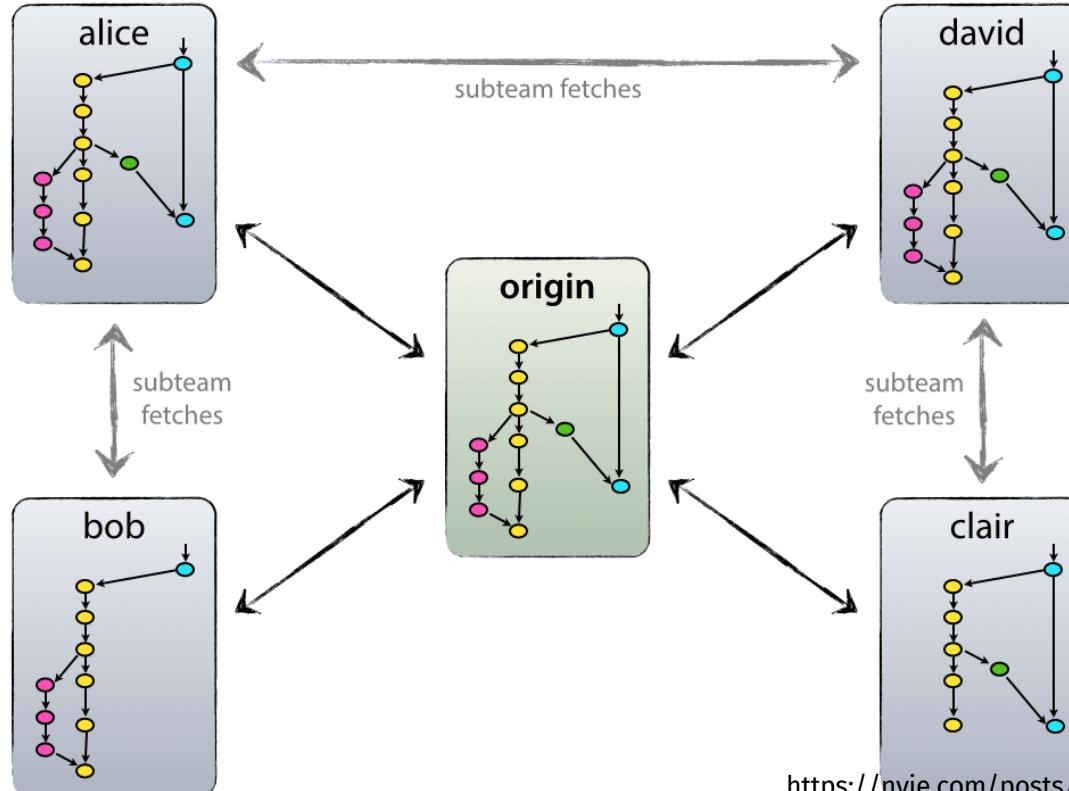
```
110 \opt{f}{\captionof{figure}{Exemple de code MainDecoder}}
111 \opt{f}{\captionof{figure}{MainDecoder Code-Beispiel}}
112 \label{fig:riscv-mainDecoder-code}
```

113
114 [subsubsection]{ALU}
115 \opt{f}{%
116 L'ALU réalise les fonctions arithmétiques et logiques selon la table suivante:
117 \begin{table}[h]
118 \caption{ALU}
119 Die ALU realisiert die arithmetischen und logischen Funktionen gemäß der folgenden Tabelle:
120 \end{table}
121 }
122 \begin{table}[h]



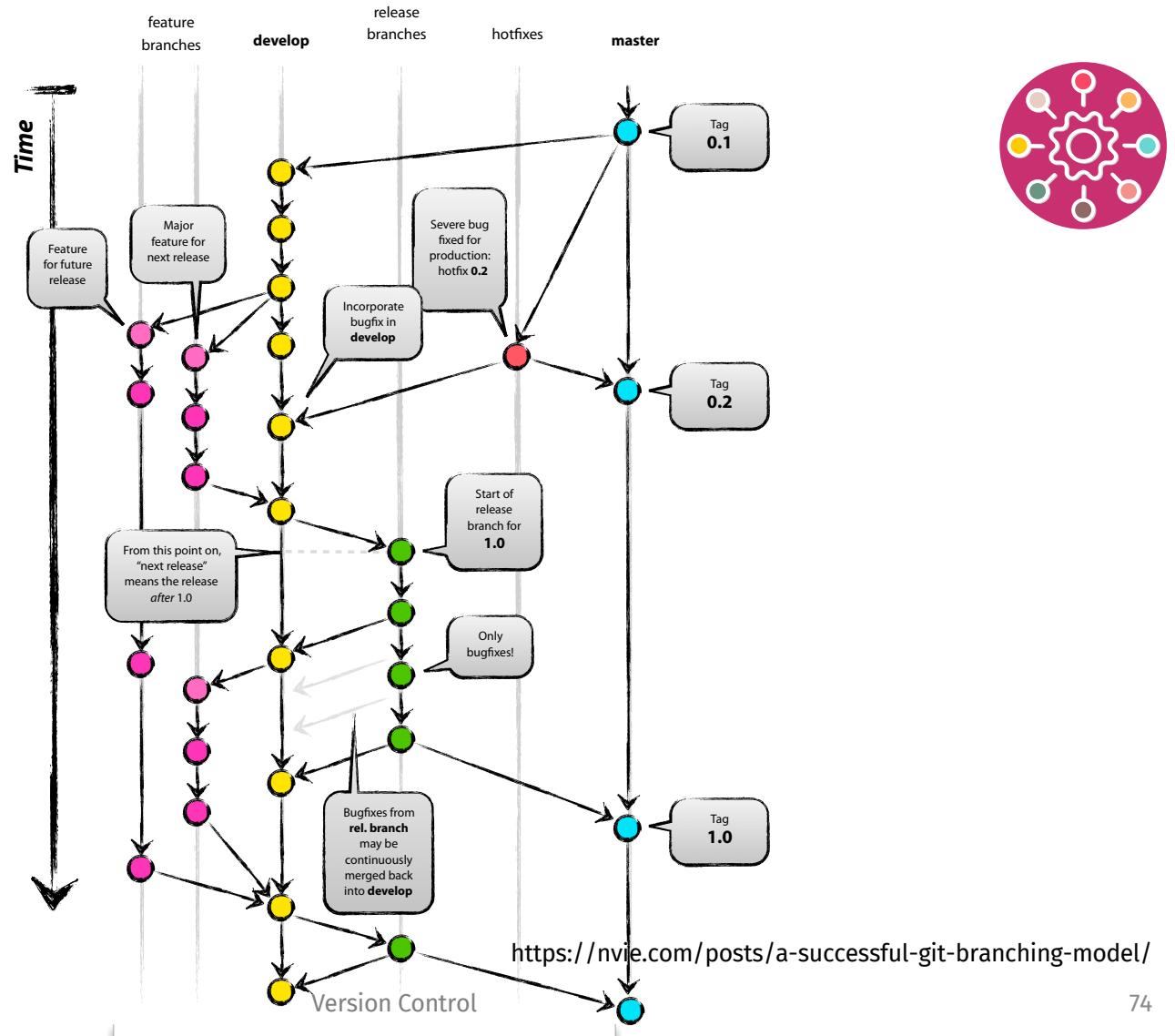
Gitflow

Gitflow Collaboration

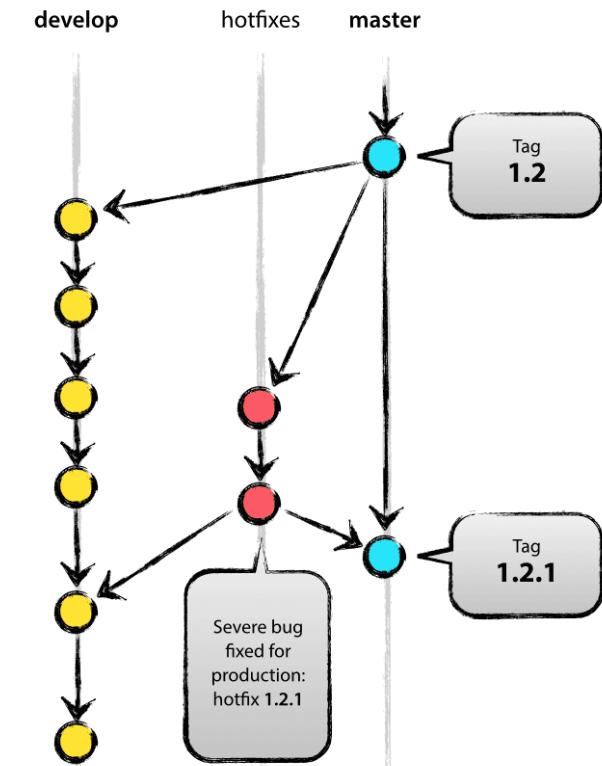
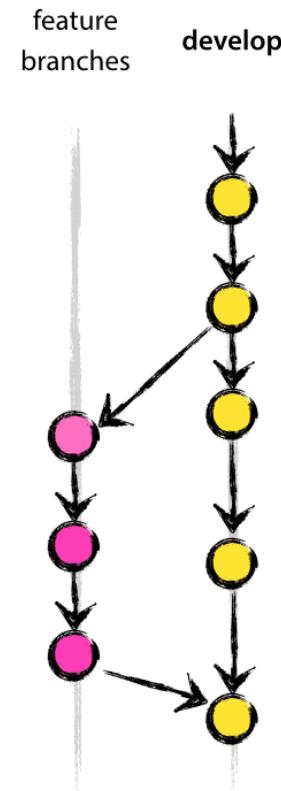
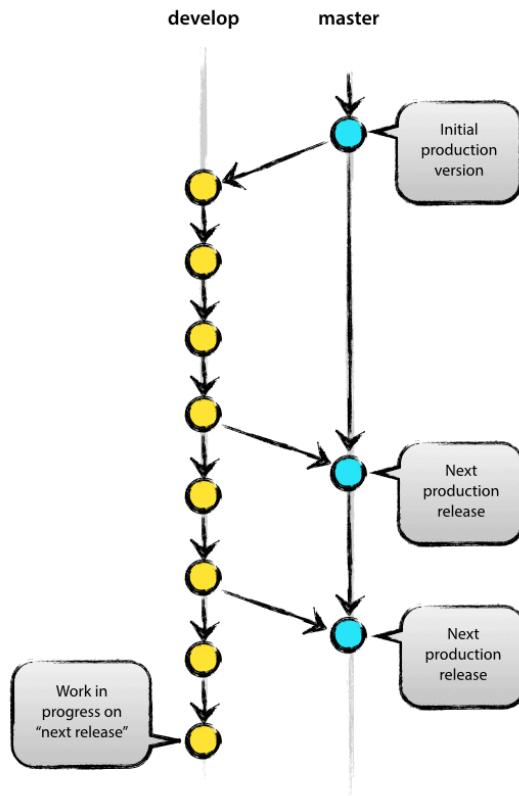


<https://nvie.com/posts/a-successful-git-branching-model/>

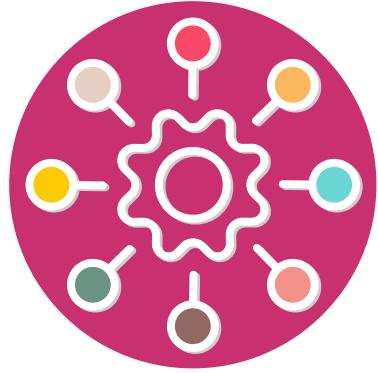
Gitflow



Gitflow Branching



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



Git CI/CD

Qu'est-ce que la CI/CD ?



- L'intégration continue (Continuous Integration, CI) est la pratique qui consiste à intégrer fréquemment des modifications de code dans un référentiel partagé, qui est ensuite automatiquement construit et testé.
- La livraison continue (Continuous Delivery, CD) va encore plus loin en déployant automatiquement les modifications du code dans des environnements de type production afin de les tester et de les valider.
- L'automatisation des tests est un élément essentiel de la CI/CD, car elle permet de détecter les bogues et autres problèmes à un stade précoce du processus de développement.
- Les outils les plus courants sont GitLab CI/CD, Github Actions, Jenkins, CircleCI et Travis CI.

Qu'est-ce que la CI/CD ?

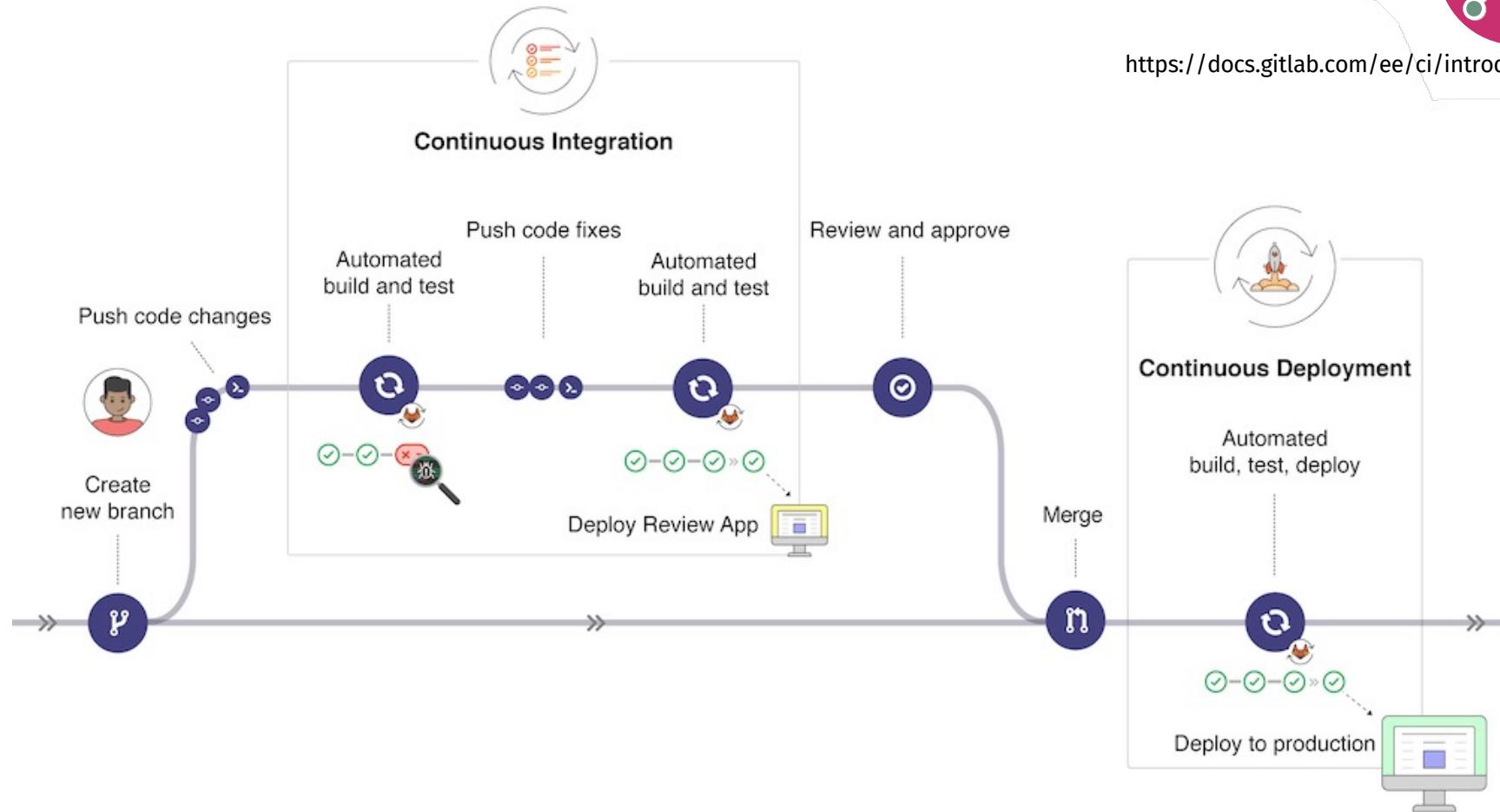


<https://about.gitlab.com/features/continuous-integration/>

Gitlab Workflow



<https://docs.gitlab.com/ee/ci/introduction/>





Estimation des tâches

Estimation 4

Créer un Star Wars
Star Destroyer (4784pcs)

X Point(s)



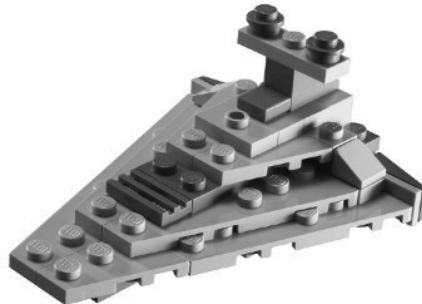
0	$\frac{1}{2}$	1	2
3	5	8	13
20	40	100	∞
?			

Estimation des tâches

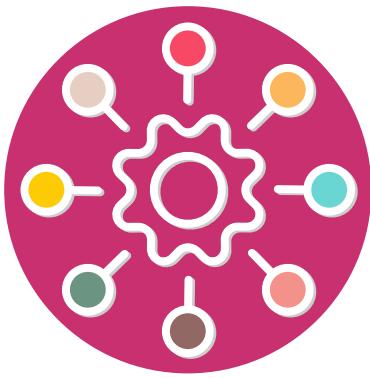
Estimation 5

Créer un Star Wars
Star Destroyer (37pcs)

X Point(s)



0	$\frac{1}{2}$	1	2
3	5	8	13
20	40	100	∞
?			



INTRODUCTION TO GIT FILE VERSIONING



Introduction to git file versioning



Contents

1 Goal	1
2 Installation	2
3 Basic operations	5
4 Branch and Merge	9
5 Gitflow	10
6 GIT commands	12