

Abschätzung von Aufgaben

Abschätzung 2

Erstelle eine kleines Haus
X Punkt(e)



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

Abschätzung von Aufgaben

Abschätzung 3

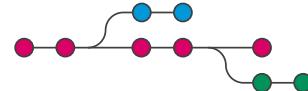
Erstelle einen
programmierbaren Rotober
X Punkt(e)



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

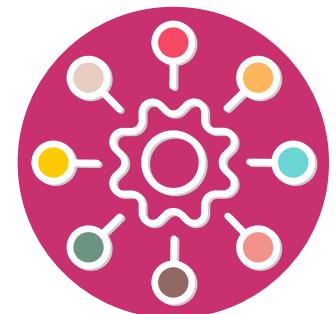


Systemdesign Version Control



Studiengang Systemtechnik

Silvan Zahno silvan.zahno@hevs.ch



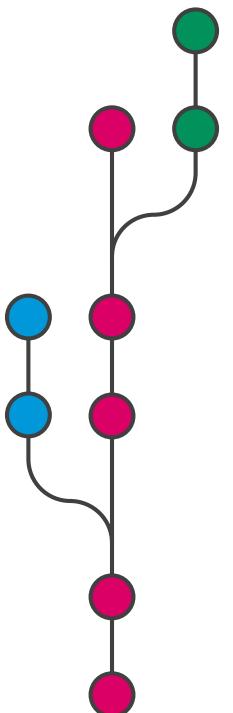


Ihr derzeitiges System

```
•
  └── wichtige-datei copy.txt
  └── wichtige-datei v1.txt
  └── wichtige-datei v2.1 backup.txt
  └── wichtige-datei v2.1 backup.txt.old
  └── wichtige-datei v2.1.txt
  └── wichtige-datei v2.txt
  └── wichtige-datei.txt
```

1 Ordner, 7 Dateien

Version control





Possible Hilfsmittel

Git (git)



Subversion (svn)

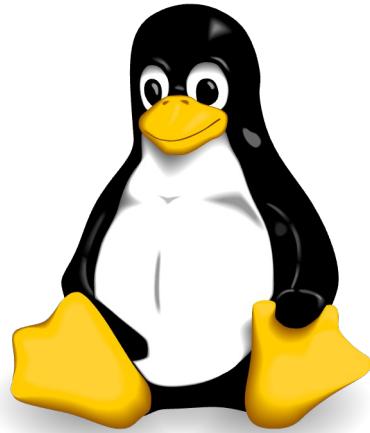


Mercurial (hg)

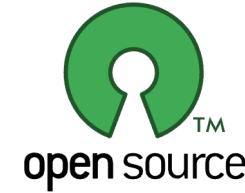


Linux Torvalds

Linux (1991)

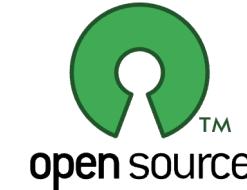
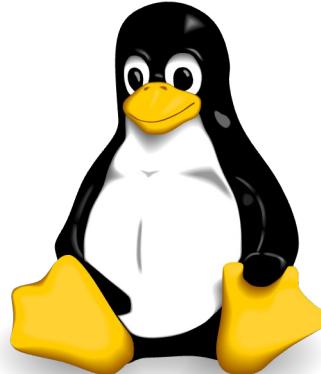


Git (2005)



Warum Linux braucht git

Linux hat sich in den letzten 30 Jahren zum grössten gemeinschaftlichen Entwicklungsprojekt in der Geschichte der Computertechnik entwickelt.



- 600 aktive Linux-Distributionen
- 85% aller Smartphones
- 500 Top-Supercomputer
- > 27,8 Millionen Codezeilen
- > 12'000 Mitwirkende
- > 1 Million commits

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

Warum für ein Power&Control oder Design&Material git notwendig ist

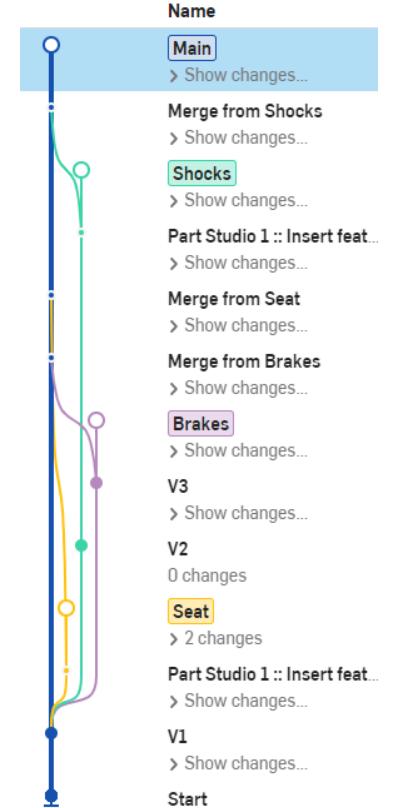


Versionierung

- Verfolgung von Änderungen
- Zusammenarbeit
- Rollback-Fähigkeiten
- Dokumentation
- Bereitstellung



AUTODESK
Vault



Git Plattformen

Gitlab

<https://gitlab.com>

<https://gitlab.hevs.ch>

Github

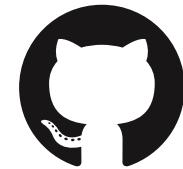
<https://github.com>

Bitbucket

<https://bitbucket.com>



GitLab



GitHub

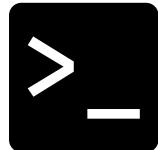


Bitbucket

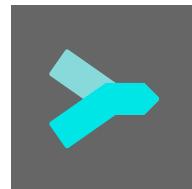


Git Hilfsmittel

Kommandozeile



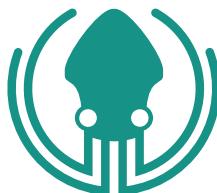
Sublime Merge



Git Cola



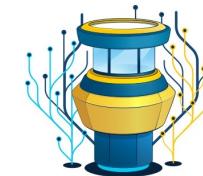
Git Kraken



Fork



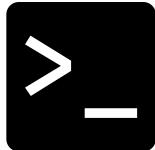
Tower



SmartGit



Git Kommandozeile



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] [-e | --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

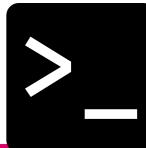
collaborate (see also: git help workflows)
fetch   Download objects and refs from another repository
pull    Fetch from and integrate with another repository or a local branch
push    Update remote refs along with associated objects

'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)
[~]$
```

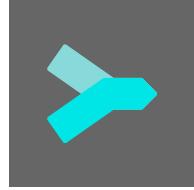


git Befehle



Befehl	Beschreibung	Befehl	Beschreibung
Eine working area beginnen		Wachsen, markieren und optimieren Sie Ihre gemeinsame Historie	
clone	Klonen eines Repositorys in ein neues Verzeichnis	branch	Zweige auflisten, erstellen oder löschen Auschecken
init	Erstellen eines leeren git-Repo oder Reinitialisieren eines vorhandenen Repo	checkout	Zweige wechseln oder Arbeitsbaumdateien wiederherstellen
An aktuellen Änderungen arbeiten		commit	Änderungen an der Projektliste aufzeichnen
add	Hinzufügen von Dateiinhalten zum Index	diff	Änderungen zwischen Commits, Commit und Arbeitsbaum anzeigen, etc.
mv	Verschieben oder Umbenennen einer Datei, eines Verzeichnisses oder eines Symlinks	merge	Zwei oder mehr Entwicklungsgeschichten zusammenführen
reset	Zurücksetzen des aktuellen HEAD auf den angegebenen Zustand	rebase	Commits auf einen anderen Basistipp anwenden
rm	Entfernen von Dateien aus dem Arbeitsbaum und aus dem Index	tag	tag
Untersuchen der Historie und des Status		Zusammenarbeiten	
log	Übergabeprotokolle anzeigen	fetch	Herunterladen von Objekten und Referenzen aus einem anderen Repo
show	Verschiedene Arten von Objekten anzeigen	pull	Abrufen und Integrieren aus einem anderen Projektarchiv oder einem lokalen Zweig schieben
status	Show the working tree status	push	Aktualisieren entfernter Referenzen zusammen mit zugehörigen Objekten

Sublime Merge



The screenshot shows a GitHub repository interface for a project named 'car-course'. The repository has the following structure:

- BRANCHES (1)**:
 - master
- REMOTES (1)**:
 - origin master
- TAGS (0)**
- STASHES (0)**
- SUBMODULES (0)**

Commits (1 untagged file, Commit Changes):

- Merge remote-tracking branch 'origin/master' (16 commits by zas, Thu, 20 Apr 08:11)
 - CHG: updated planning (4 commits by zas, Thu, 13 Apr 15:19)
 - ADD: ALU and immSrc doc (4 commits by Axam, Thu, 13 Apr 15:19)
 - ADD: EBS2/EBS3 specs (4 commits by Axam, Tue, 11 Apr 15:25)
 - FIX: memory stack images (5 commits by zas, Tue, 4 Apr 11:00)
 - FIX: errors in immediate and type images (44 commits by zas, Tue, 4 Apr 07:46)
 - ADD: files in arc exercises (33 commits by zas, Mon, 3 Apr 13:30)
 - ADD: note on Ripes memory management (11 commits by Axam, Fri, 31 Mar 15:38)
 - CHG: Planning (4 commits by zas, Fri, 31 Mar 08:45)
 - FIX: reverse engineering solution (8 commits by Axam, Thu, 30 Mar 17:26)
 - FIX: ISA syntax errors (3 commits by zas, Tue, 28 Mar 07:59)
 - Merge remote-tracking branch 'origin/master' (6 commits by zas, Tue, 28 Mar 07:29)
 - FIX: errors in ISA (22 commits by zas, Tue, 28 Mar 07:29)
 - ADD: windows Geekbench window (5 commits by Axam, Thu, 16 Mar 10:14)
 - FIX: add scripts folder (Axel Amand, Thu, 16 Mar 09:31)
 - REM: car-hdrv and car-labs doc deployment (Axel Amand, Thu, 16 Mar 09:18)
 - CHG: BEM labo from geekbench 5 to 6 (14 commits by zas, Tue, 14 Mar 14:25)
 - UPD: all PDFs (30 commits by Axam, Wed, 8 Mar 19:10)
 - FIX: errors in ARC, ISA, FUN and PER slides (26 commits by zas, Tue, 7 Mar 09:09)

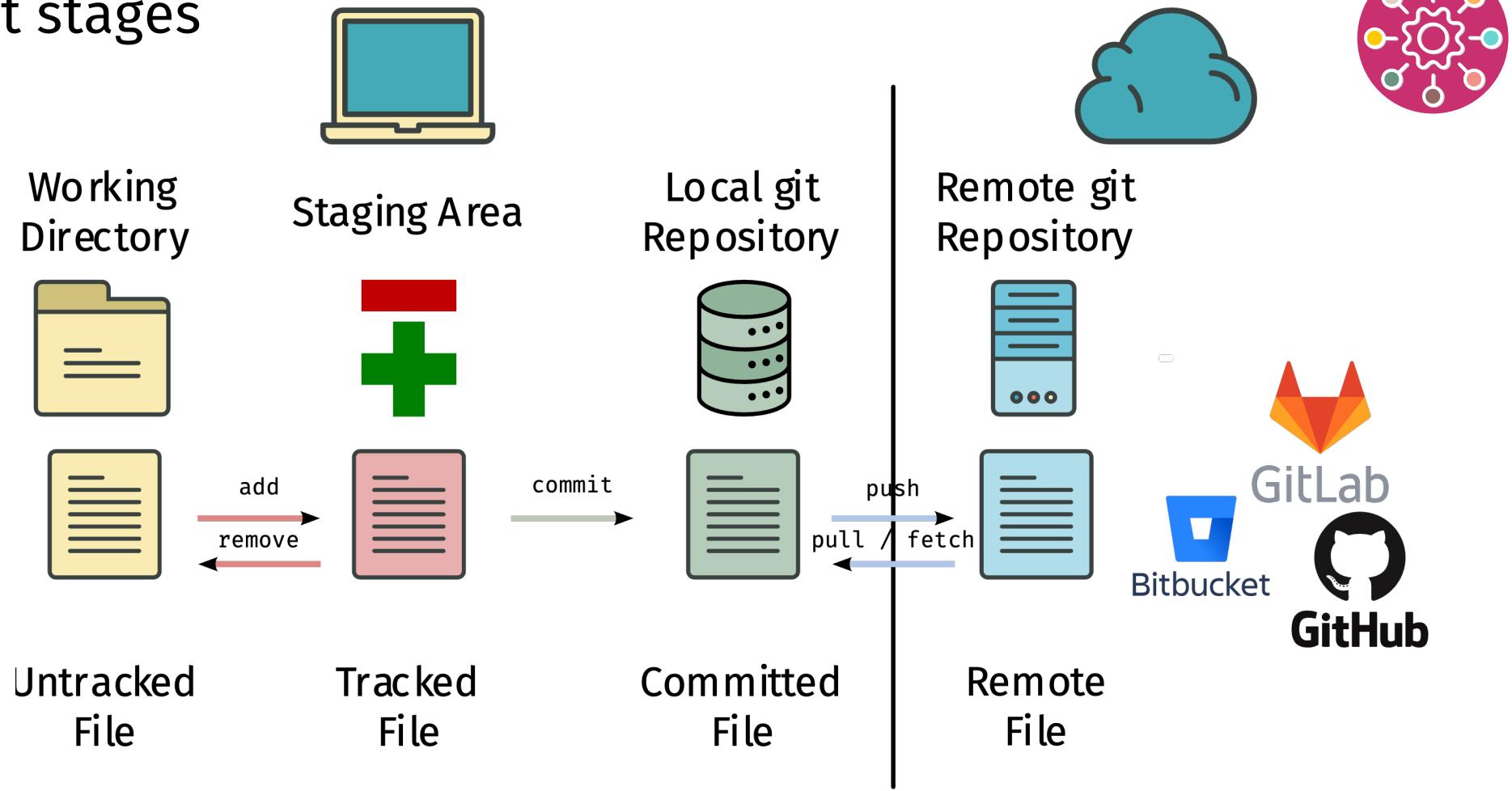
Files (Summary):

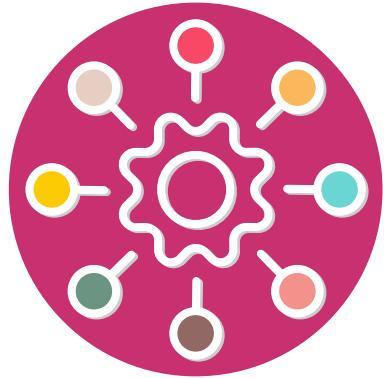
- Commit Hash: 702cb1f738add663984a48b72e4bc68361d3c
- Tree: f163ceab55f992b7c7854993994b8676e0da8b
- Author: zas <silvan.zahn@hevs.ch>
- Date: Thu, 20 Apr 2023 08:12
- Parents: 6e927afe, 68810079
- Branches: master origin/master
- Stats: 16 files changed: -15 +10

Code Review (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):

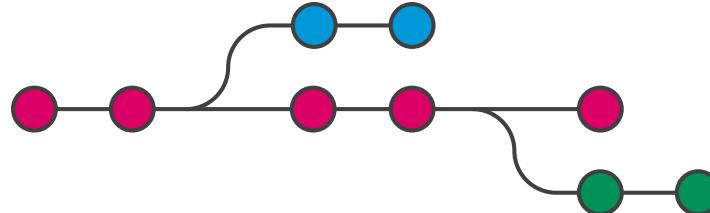
- 00-solution.tex**:
 - Subsection: Simulation:
 - 164 done:
165 beq x2, x2, main # infinite loop
166 end;#intended;
167 Each instruction takes one clock cycle => 19 are executed (addi x5, x0, 0 not executed because of previous beq; addi x2, x0, 1 not executed because of jal).
=> 19*60M = 287.9 ns.
- 03-controlunit.tex**:
 - Subsection: Umsetzung:
 - 164 Le [textbf{mainDecoder}] peut être écrit en VHDL. Pour cela, vous pouvez analyser l'exemple de code [ref:#riscv-mainDecoder-code] ci-dessous et l'adapter en conséquence.
165 **inabox**[Dans HDL Designer, lorsque vous sélectionnez le type de contenu d'un bloc, choisissez [textbf{VHDL File} => Architecture], et contrôlez que le langage est défini sur [textbf{VHDL 2008}]. Sur la page suivante, [textbf{Architecture}] correspond au nom de la vue (un bloc peut avoir différents contenus) et [textbf{Entity}] au nom du bloc (mainDecoder par exemple).]
166 **inabox**[Dans HDL Designer, lorsque vous sélectionnez le type de contenu d'un bloc, choisissez [textbf{VHDL File} => Architecture], et contrôlez que le langage soit défini sur [textbf{VHDL 2008}]. Sur la page suivante, [textbf{Architecture}] correspond au nom de la vue (un bloc peut avoir différents contenus) et [textbf{Entity}] au nom du bloc (mainDecoder par exemple).]
167 ...
168 **inabox**[
169 Schreiben Sie hierzu für beide Subblöcke, [textbf{mainDecoder}] sowie [textbf{ALUDecoder}], eine Wahrheitstabelle für alle benötigten Instruktionen.
170 **inoptf**(
171 Caption of figure: Exemple de code MainDecoder)
172 Caption of figure: MainDecoder Code-Beispiel)
173 Label:fig:riscv-mainDecoder-code)
174 **subsubsection**[ALU]
175 **inoptf**(
176 L'ALU réalise les fonctions arithmétiques et logiques selon la table suivante:
177 Caption of figure:
178 Caption of figure:
179 Label:fig:riscv-mainDecoder-code)
180 }
181 ...
182 **begin**(table)[h]

Git stages





Git Branch und Merge Beispiele

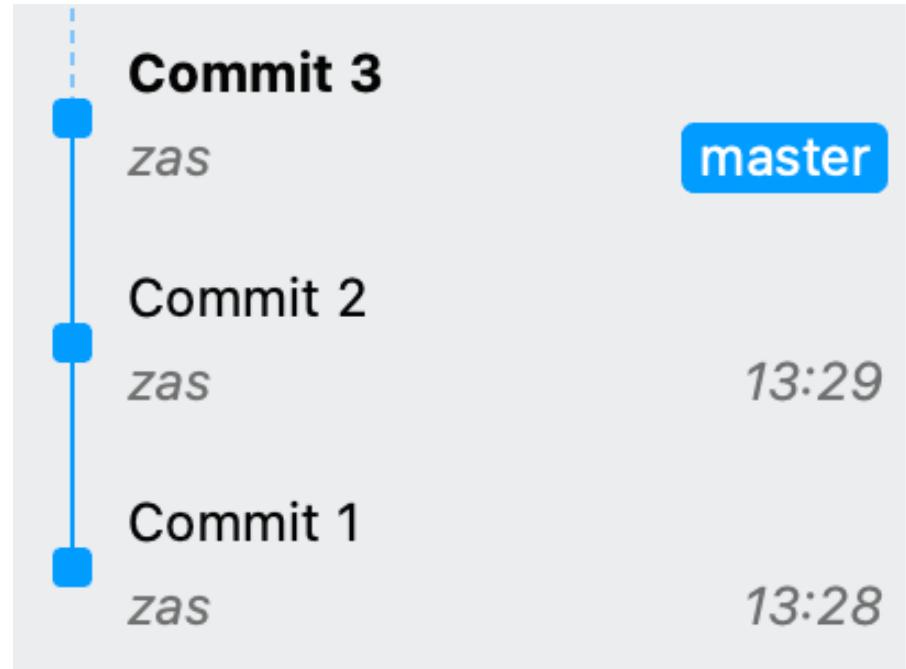




Branch und Merge

Initial repo state

Jedes repo hat entweder ein **main** oder eine **master** branch als standart branch



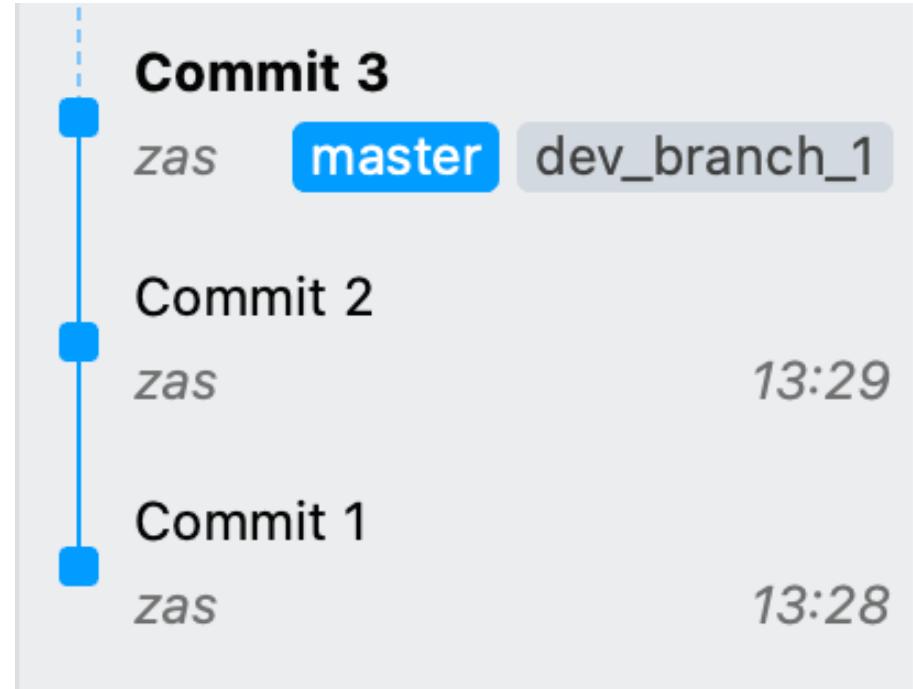


Branch und Merge

Create branch dev_branch_1

Erstelle branch dev_branch_1

```
$ git branch dev_branch_1
```





Branch und Merge

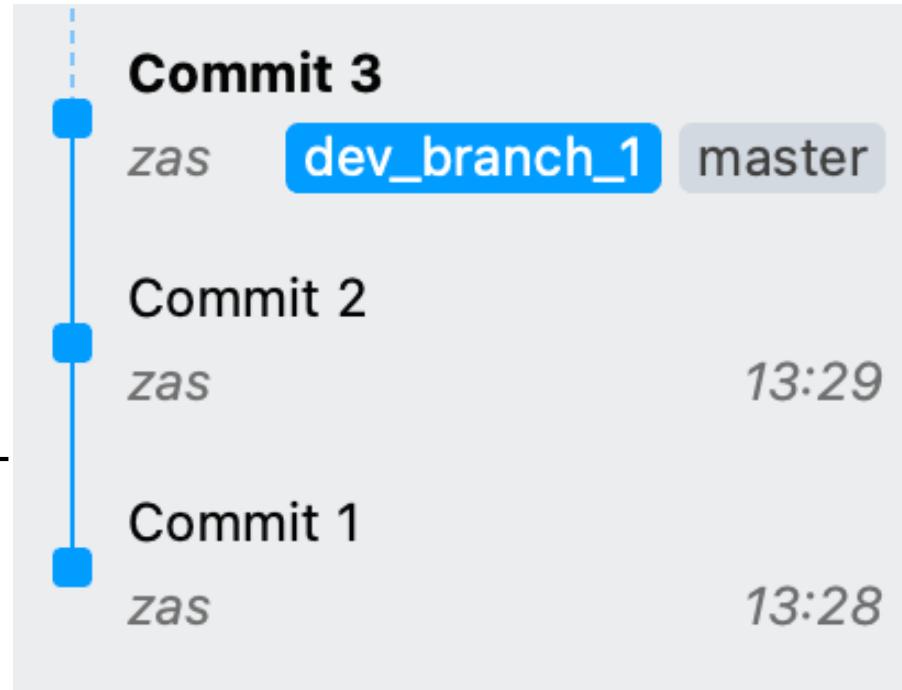
Checkout branch dev_branch_1

Prüfen Sie, in welchem Zweig wir uns befinden

```
$ git branch
```

Checkout branch dev_branch_1

```
$ git checkout dev_branch_1
```





Branch and Merge

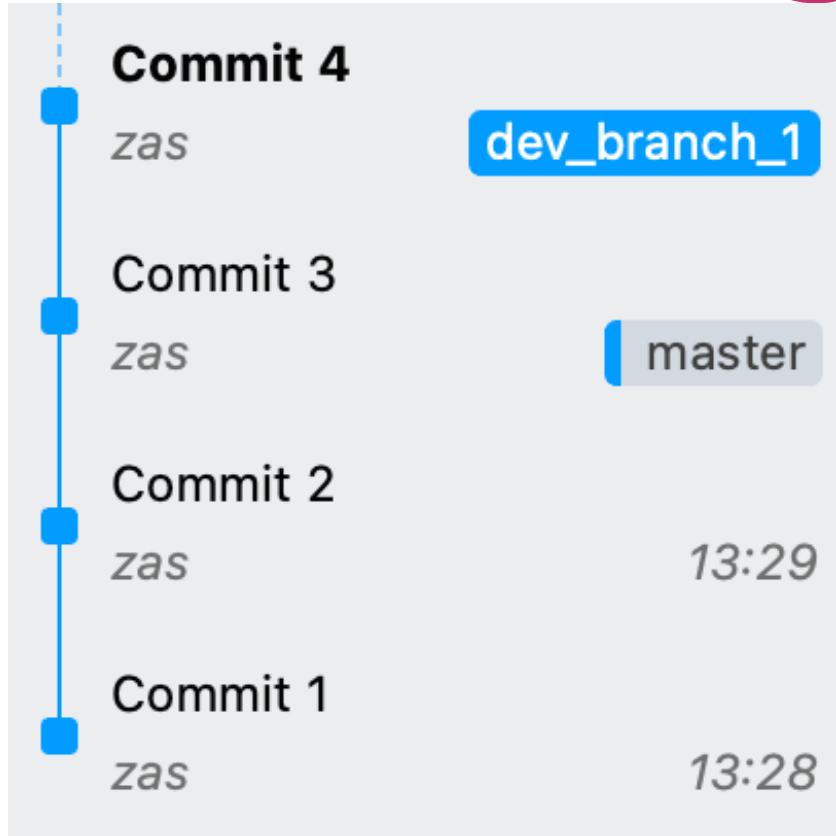
Commit on dev_branch_1

Stage new file

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

Commit stages files

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





Branch und Merge

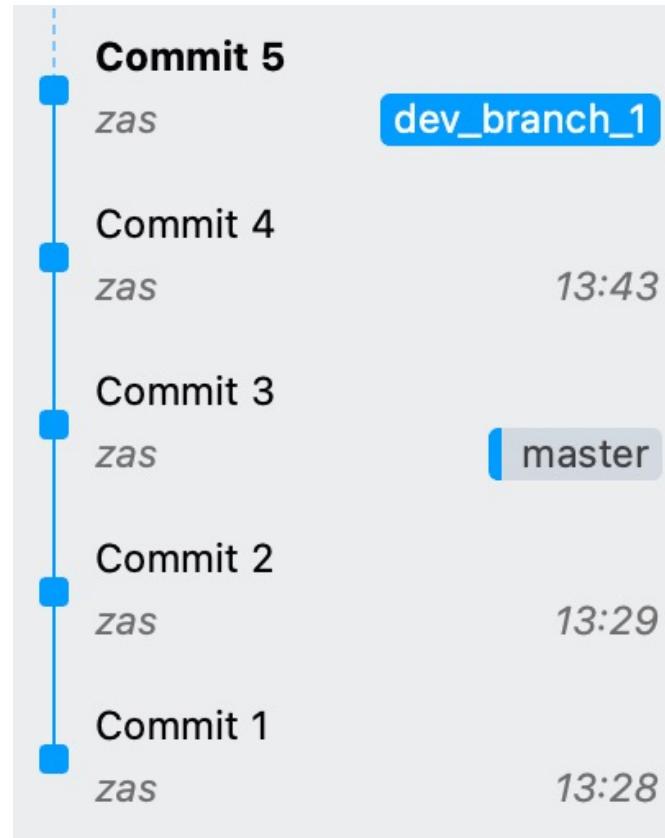
Commit on dev_branch_1

Stage new file

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

Commit stages files

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





Branch und Merge

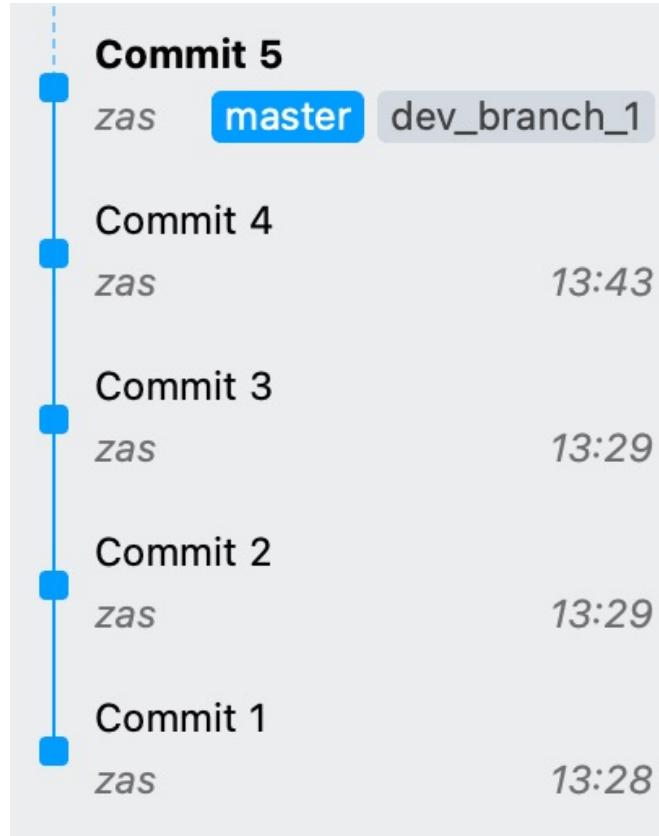
Merge master and dev_branch_1

Checkout master branch

```
$ git checkout master
```

Merge dev_branch_1 into master

```
$ git merge dev_branch_1
```





Branch und Merge

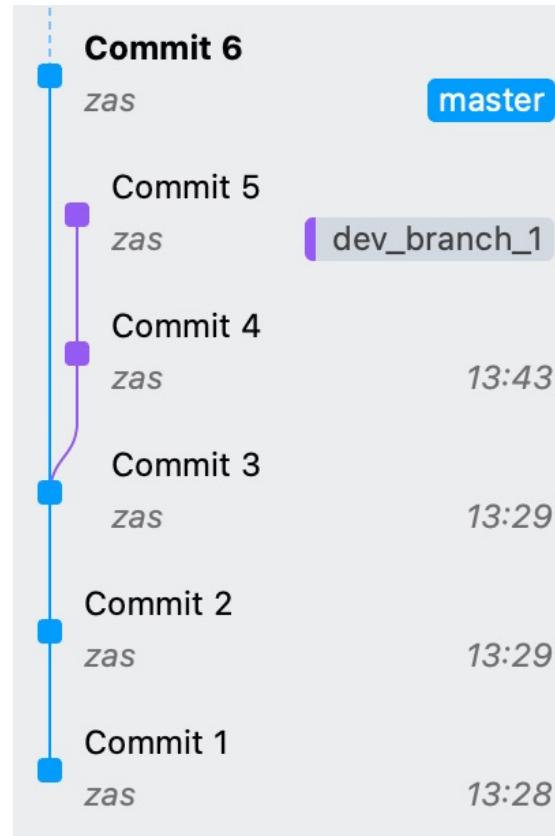
Commit on master branch

Stage new file

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

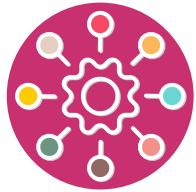
Commit stages files

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



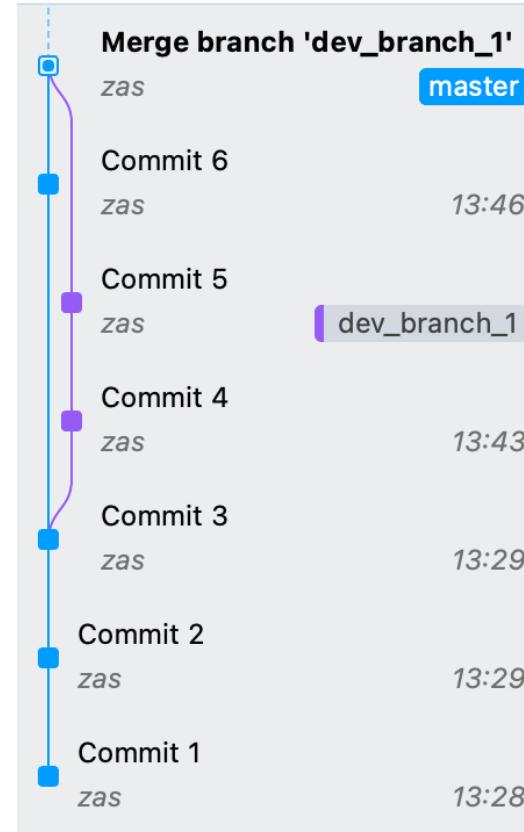
Branch und Merge

Three way merge master and dev_branch_1



Merge dev_branch_1 into master

```
$ git merge dev_branch_1
```



Demonstration



~/work/repo/edu/car/car-course

LICENSE UPGRADE REQUIRED

Locations Commits Files Summary

BRANCHES (1) master

REMOTES (1) origin

TAGS (0)

STASHES (0)

SUBMODULES (0)

Commit Hash: 702c8f1738addfb639884c48b724bcb8361d3c
Tree: f163cea5c5f992927c7854993694b8676e0da8b
Author: zas <silvan.zaho@hevs.ch>
Date: Thu, 20 Apr 2023 08:12
Parents: 6e927efb, 68810079
Branches: master, origin/master
Stats: 16 files changed: 16 +10

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

CHG: updated planning (1 commit)

ADD: ALU and ImmSrc doc (1 commit)

ADD: EBS2/EBS3 specs (1 commit)

FIX: memory stack images (1 commit)

FIX: errors in immediate and type images (44 commits)

ADD: files in arc exercises (33 commits)

ADD: note on Ripes memory management (11 commits)

CHG: Planning (4 commits)

FIX: reverse engineering solution (8 commits)

FIX: ISA syntax errors (3 commits)

Merge remote-tracking branch 'origin/master' (6 commits)

FIX: errors in ISA (5 commits)

ADD: windows Geekbench window (5 commits)

FIX: add scripts folder (1 commit)

REM: car-heirv and car-labs doc deployment (1 commit)

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

UPD: all PDFs (30 commits)

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

Summary: 00-solution.tex, 03-controllunit.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

Commit Hash: 702c8f1738addfb639884c48b724bcb8361d3c
Tree: f163cea5c5f992927c7854993694b8676e0da8b
Author: zas <silvan.zaho@hevs.ch>
Date: Thu, 20 Apr 2023 08:12
Parents: 6e927efb, 68810079
Branches: master, origin/master
Stats: 16 files changed: 16 +10

Merge remote-tracking branch 'origin/master'

Subsection: Simulation:

```
164 done;
165    beq x2, x2, main      # infinite loop
166 }end(minted)
167 \newline\newline\newline
168 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).
=> 19/60M = 287.9 ns.
```

Subsection: Simulation:

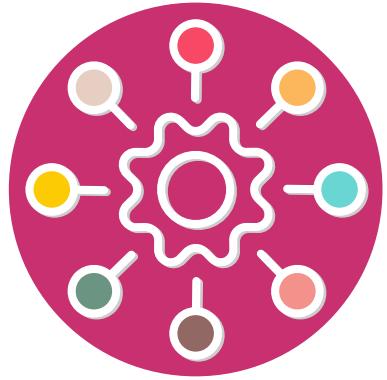
```
164 done;
165    beq x2, x2, main      # infinite loop
166 }end(minted)
167 \newline\newline\newline
168 Each instruction takes one clock cycle => 19 are executed (addi x5, x0, 0 not
executed because of previous beq; addi x2, x0, 1 not executed because of jal).
169 On the EBS2 board @ 60MHz \rightarrow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

Subsection: Umsetzung:

```
164 Le \textbf{mainDecoder} peut être écrit en VHDL. Pour cela, vous pouvez analyser
l'exemple de code \ref{fig:riscv-mainDecoder-code} ci-dessous et l'adapter en
conséquence.
165
166 \notebox{Dans HDL Designer, lorsque vous sélectionnez le type de contenu d'un
bloc, choisissez \textbf{VHDL File -> Architecture}, et contrôlez que le
langage est défini sur \textbf{VHDL 2008}. Sur la page suivante, \textbf{Architecture}
correspond au nom de la vue (un bloc peut avoir différents
contenus) et \textbf{Entity} au nom du bloc (\textbf{mainDecoder} par exemple).}
167 }
168 \opt{d}{%
169 Schreiben Sie hierzu für beide Subblöcke, \textbf{mainDecoder} sowie \textbf{ALUDecoder},
eine Wahrheitstabelle für alle benötigten Instruktionen.
170 }
```

Subsection: Umsetzung:

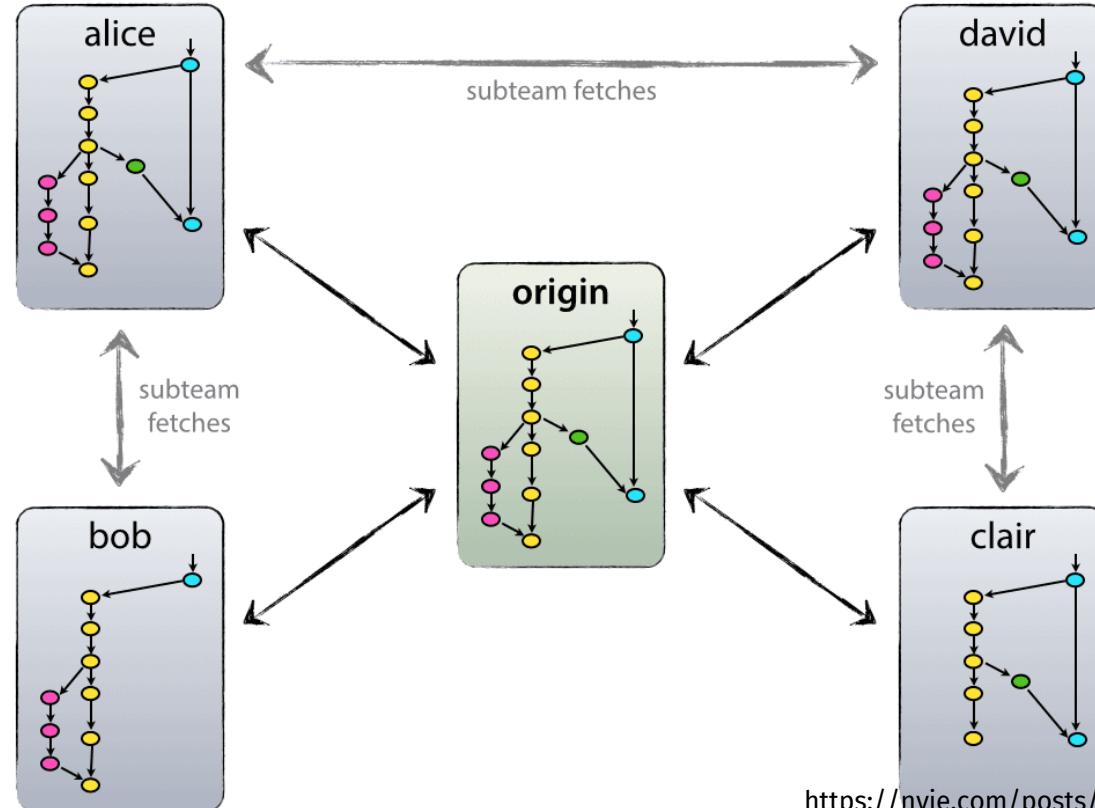
```
164 Le \textbf{mainDecoder} peut être écrit en VHDL. Pour cela, vous pouvez analyser
l'exemple de code \ref{fig:riscv-mainDecoder-code} ci-dessous et l'adapter en
conséquence.
165
166 \notebox{Dans HDL Designer, lorsque vous sélectionnez le type de contenu d'un
bloc, choisissez \textbf{VHDL File -> Architecture}, et contrôlez que le
langage soit défini sur \textbf{VHDL 2008}. Sur la page suivante, \textbf{Architecture}
correspond au nom de la vue (un bloc peut avoir différents
contenus) und \textbf{Entity} au nom du bloc (\textbf{mainDecoder} par exemple).}
167 }
168 \opt{d}{%
169 Schreiben Sie hierzu für beide Subblöcke, \textbf{mainDecoder} sowie \textbf{ALUDecoder},
eine Wahrheitstabelle für alle benötigten Instruktionen.
170 }
171 \begin{center}
172 \textbf{fig:riscv-mainDecoder-code:}
173 \begin{lstlisting}[language=VHDL]
174 \opt{f}{\captionof{figure}{Exemple de code MainDecoder}}
175 \opt{d}{\captionof{figure}{MainDecoder Code-Beispiel}}
176 \label{fig:riscv-mainDecoder-code}
177 \end{lstlisting}
178 \end{center}
179
180 \subsubsection{ALU}
181 \opt{f}{%
182 L'ALU réalise les fonctions arithmétiques et logiques selon la table suivante:
183 }
184 \opt{d}{%
185 Die ALU realisiert die arithmetischen und logischen Funktionen gemäß der folgenden
Tabelle:
186 }
187
188 \begin{table}[h]
```



Gitflow

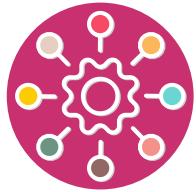
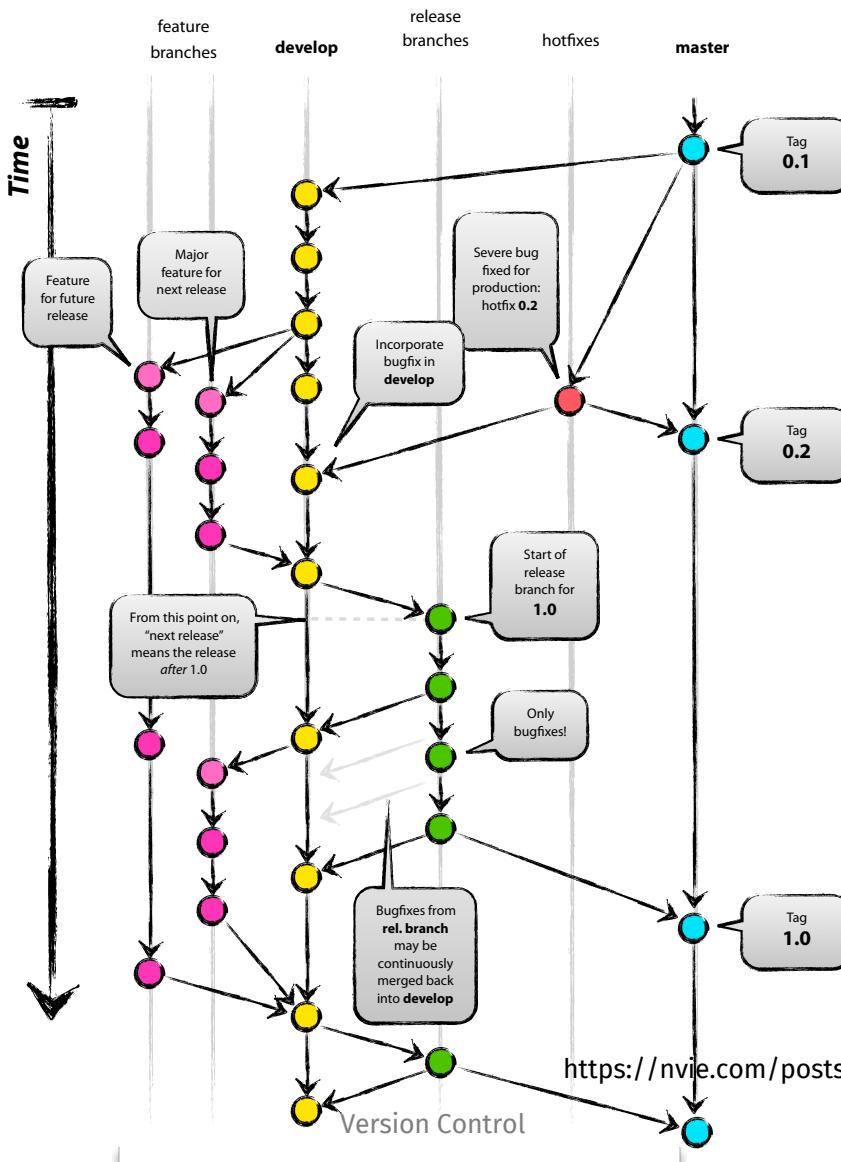


Gitflow Zusammenarbeit

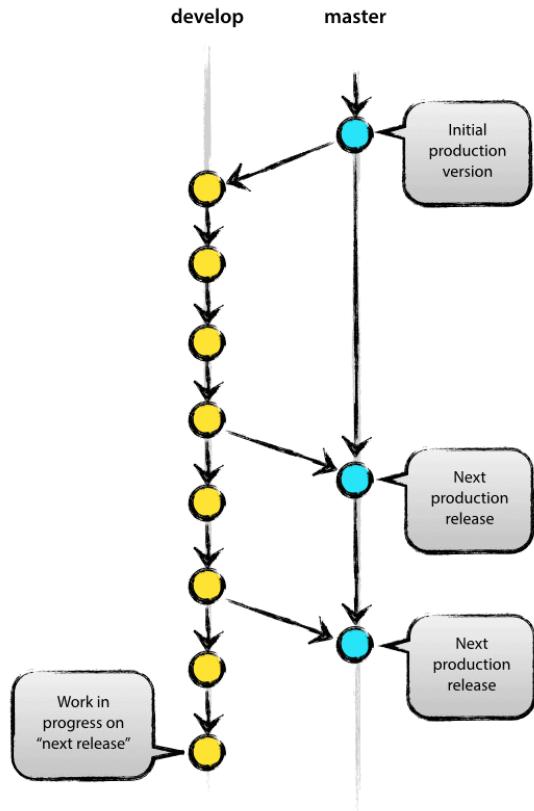


<https://nvie.com/posts/a-successful-git-branching-model/>

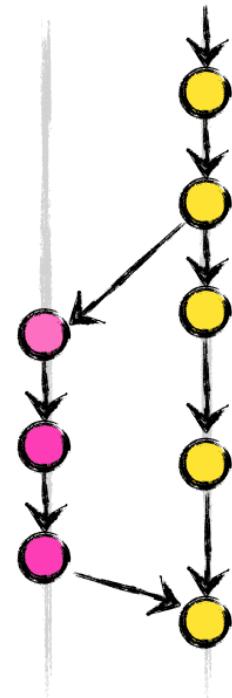
Gitflow



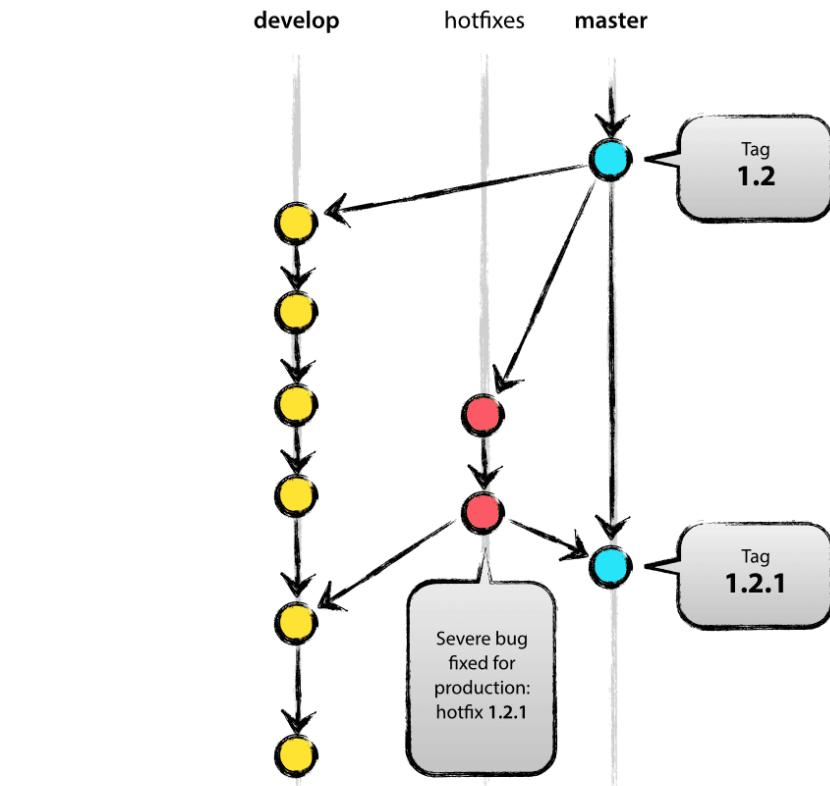
Gitflow Branching



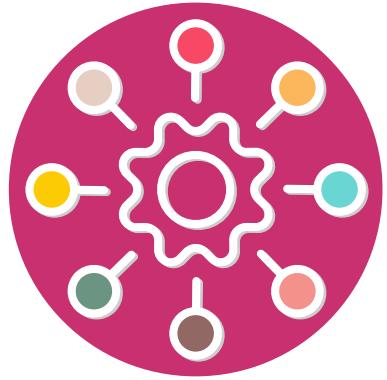
feature
branches



develop



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



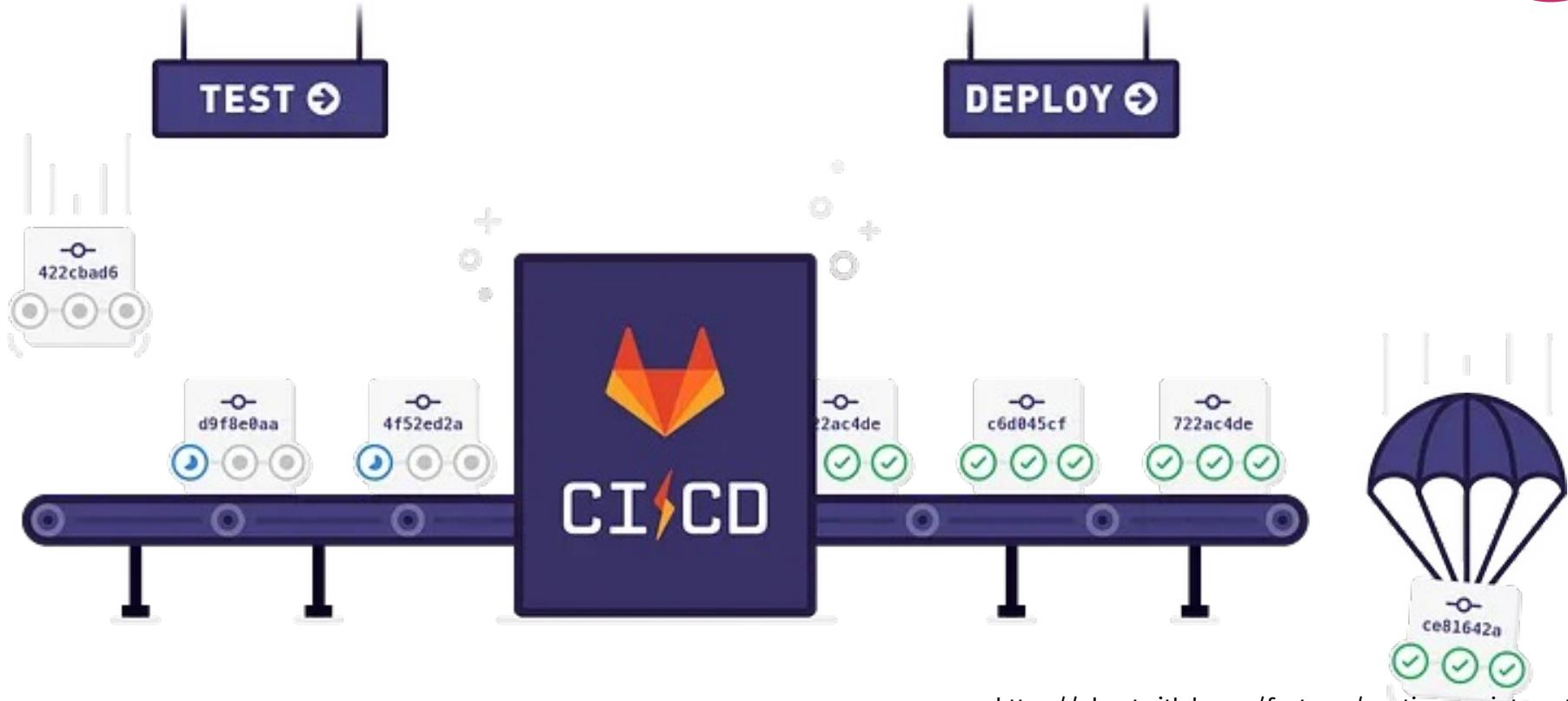
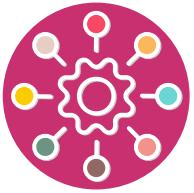
Git CI/CD



Was ist CI/CD?

- Bei der kontinuierlichen Integration (Continuous Integration, CI) werden Codeänderungen häufig in ein gemeinsames Repository integriert, das dann automatisch erstellt und getestet wird.
- Continuous Delivery (CD) geht noch einen Schritt weiter, indem Codeänderungen automatisch in produktionsähnlichen Umgebungen für weitere Tests und Validierungen bereitgestellt werden.
- Automatisierte Tests sind eine wichtige Komponente von CI/CD, da sie dazu beitragen, Bugs und andere Probleme frühzeitig im Entwicklungsprozess zu erkennen.
- Beliebte Tools sind GitLab CI/CD, Github Actions, Jenkins, CircleCI und Travis CI.

Was ist CI/CD?

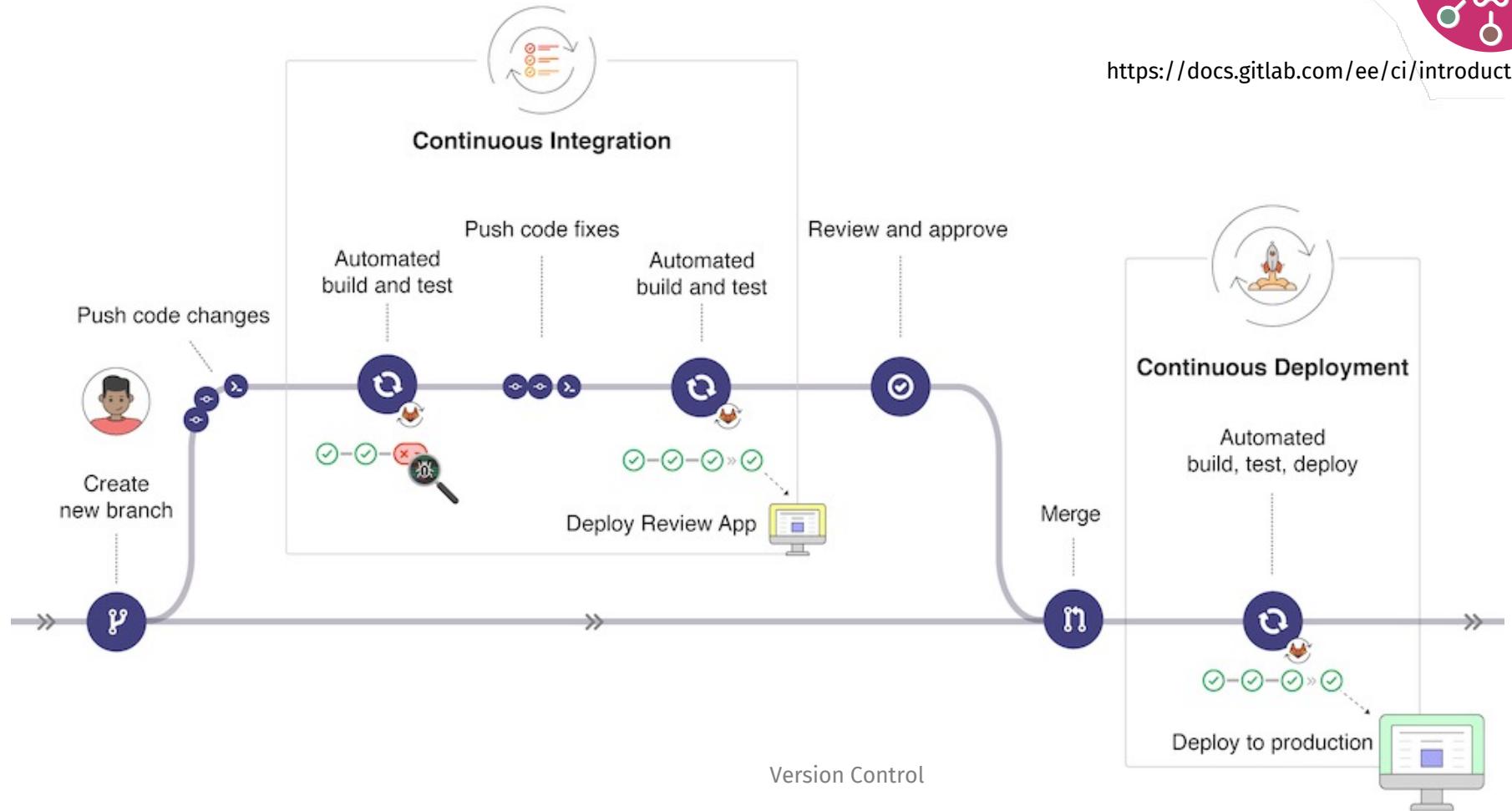


<https://about.gitlab.com/features/continuous-integration/>

Gitlab Workflow



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



Abschätzung von Aufgaben

Abschätzung 4

Erstelle einen Star Wars
Star Destroyer (4784pcs)

X Punkt(e)

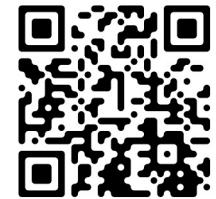


SyD Technical Management

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

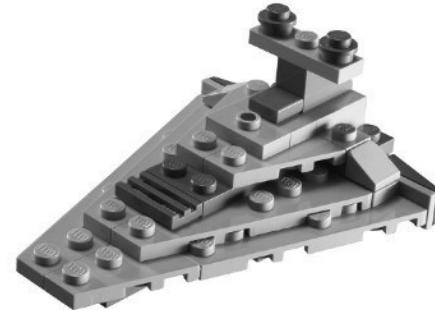
Abschätzung von Aufgaben

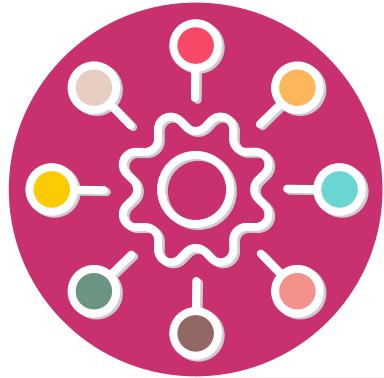
Abschätzung 5



Erstelle einen Star Wars
Star Destroyer (37pcs)

X Punkt(e)





SYSTEMS DESIGN / INTRODUCTION TO GIT - PART A

Introduction to git - Part A

Installation & Setup



SYSTEMS DESIGN / INTRODUCTION TO GIT - PART B

Introduction to git - Part B



Contents

1 Goal	1
2 Installation	2
3 Markdown	5
4 Outro	7
A GIT commands	8
B Most used Git commands	9

Contents

1 Goals	2
2 Outils	2
3 Basis Operationen	3
4 Branch and Merge	11
5 Gitgraph	15
6 Gitflow	16
7 Extras	18