
F*** YOURSELF
with GIT

Disclaimer

Eigentlich hab ich überhaupt keine Ahnung von dem was ich hier gerade rede. Sei es drum, ich wurde durch die Mitarbeit in einem Open Source Projekt dazu gezwungen, git zu verwenden.

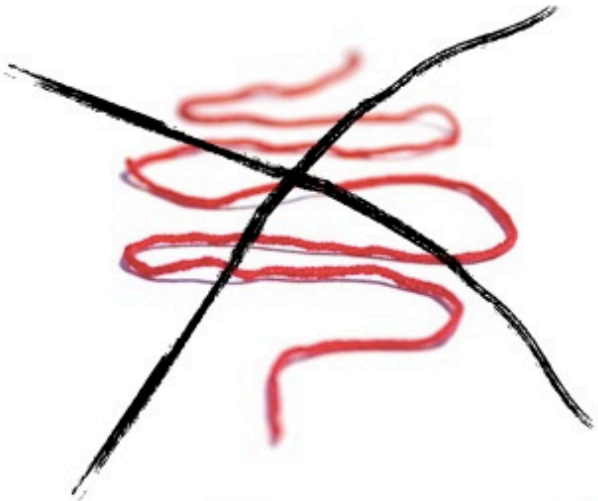
sigh Die Welt war doch so schön mit SVN!

Auberdem ist das eine gute Gelegenheit diese ganzen blöden Animationen hier mal auszuprobieren. Die Präsentation steckt voller Copyright violations. Ich bin ein Opfer der Google Bildersuche.

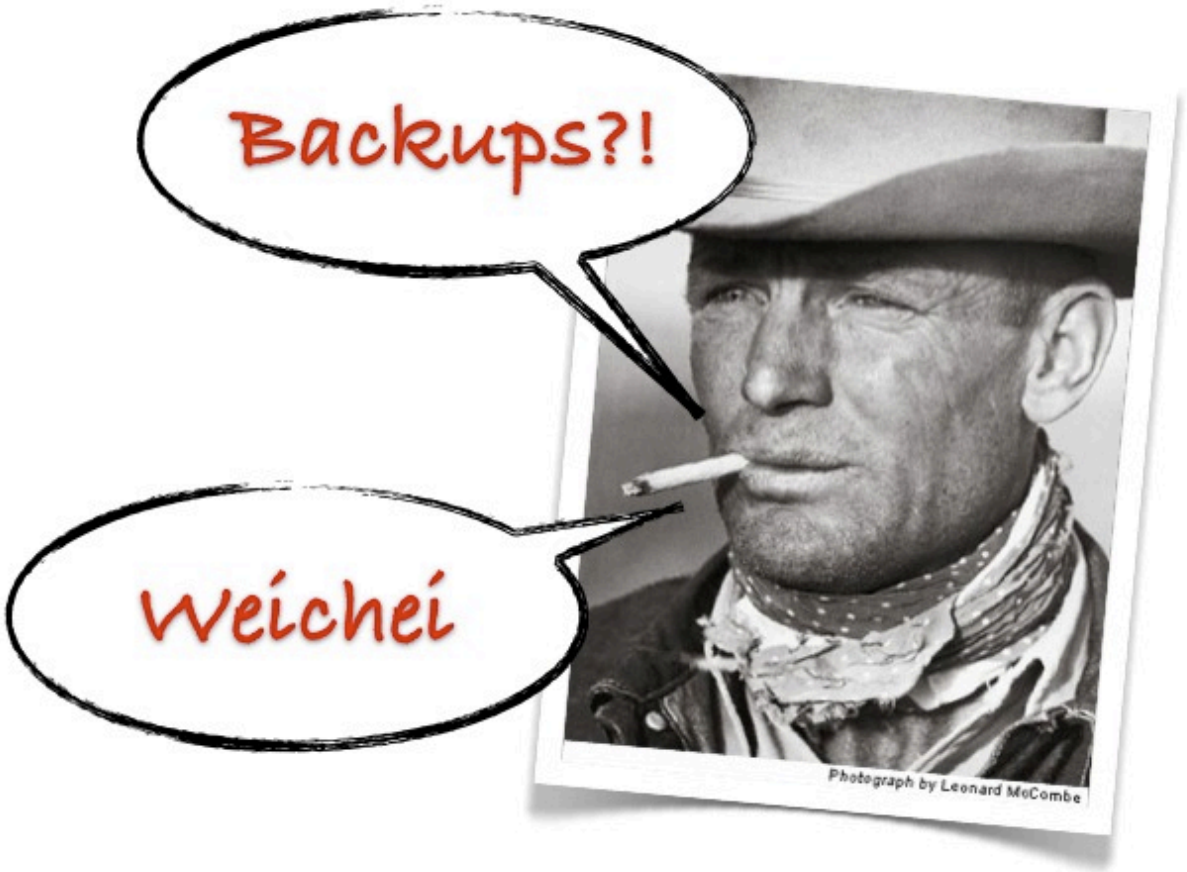
Versionskontrolle ist was für Hustensaftlutscher!

FINISH HIM!! SVN vs. GIT

GIT ist total einfa ARRRGGH!



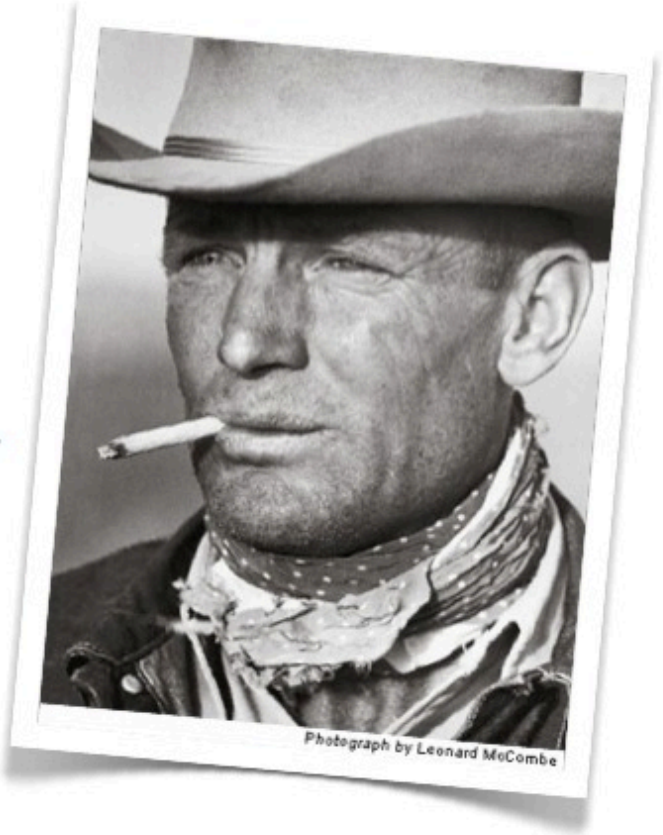
Scheisse! Ohne google gehts net?!

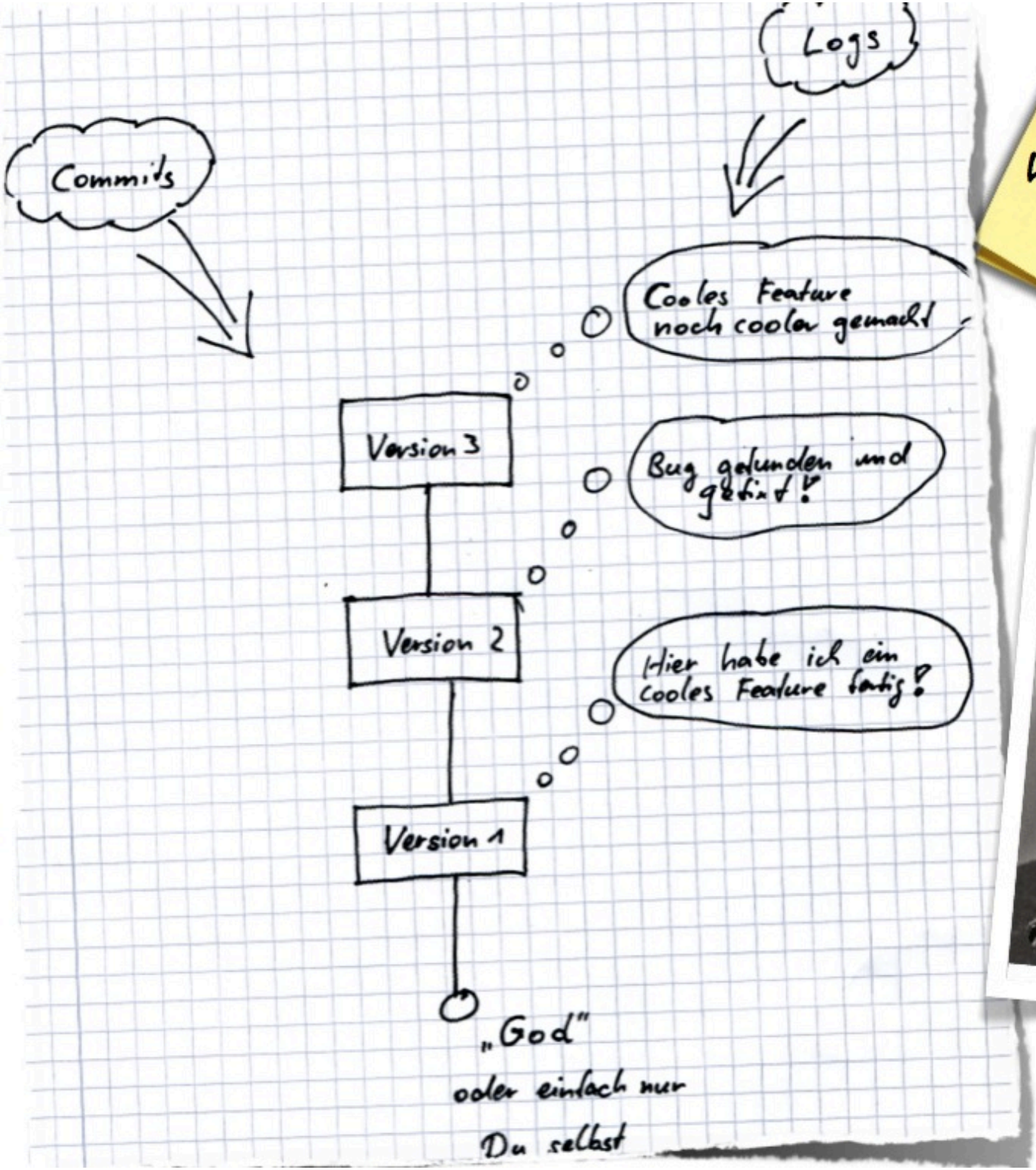

Backups?!
Weichei
Photograph by Leonard McCombe

Protokollieren

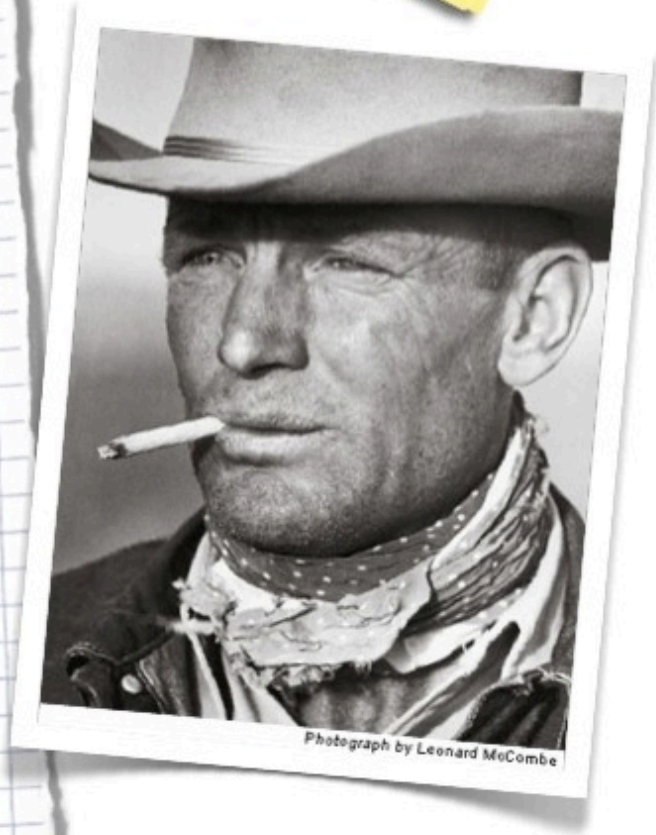
Archivieren

Wiederherstellen

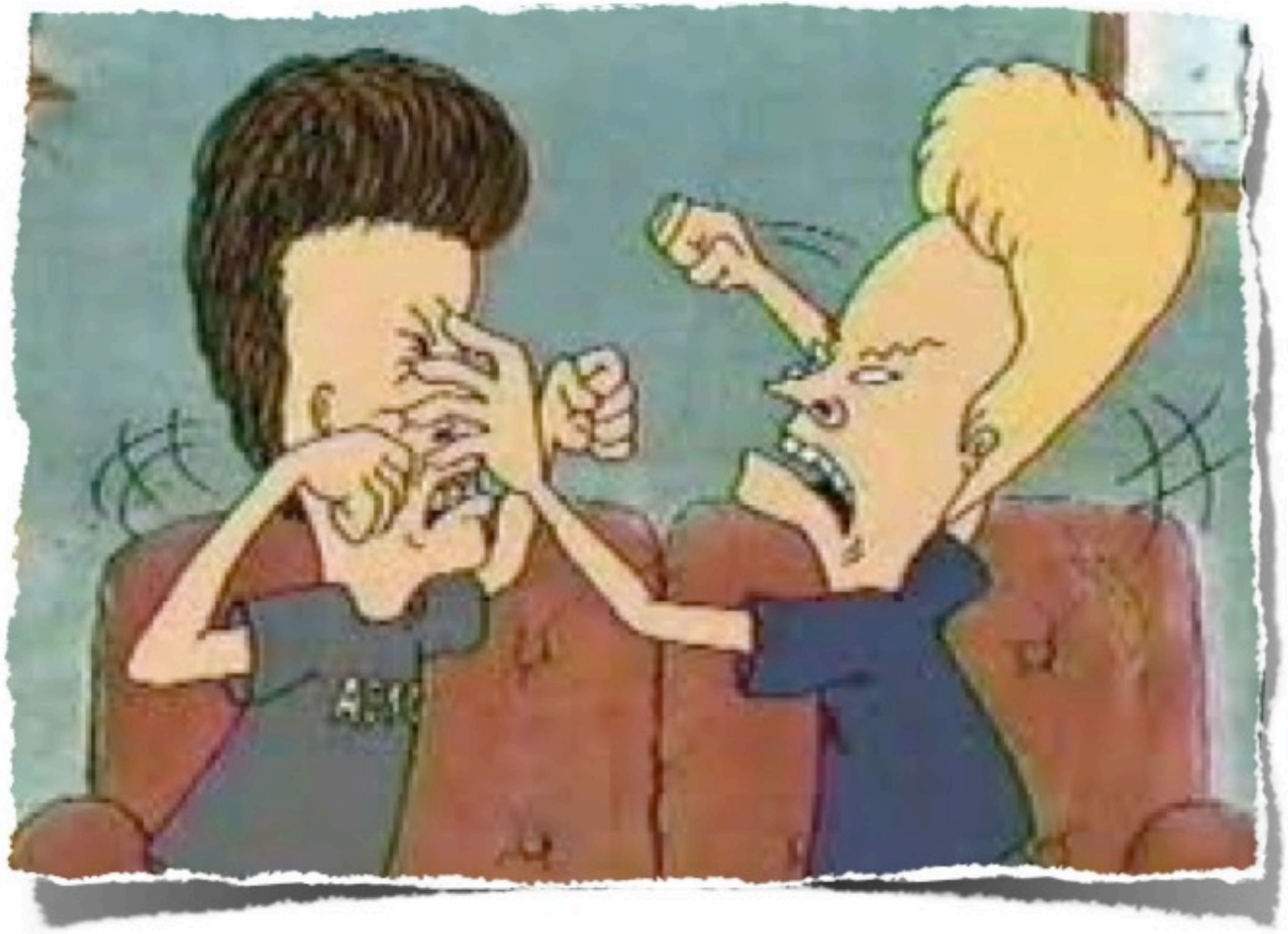




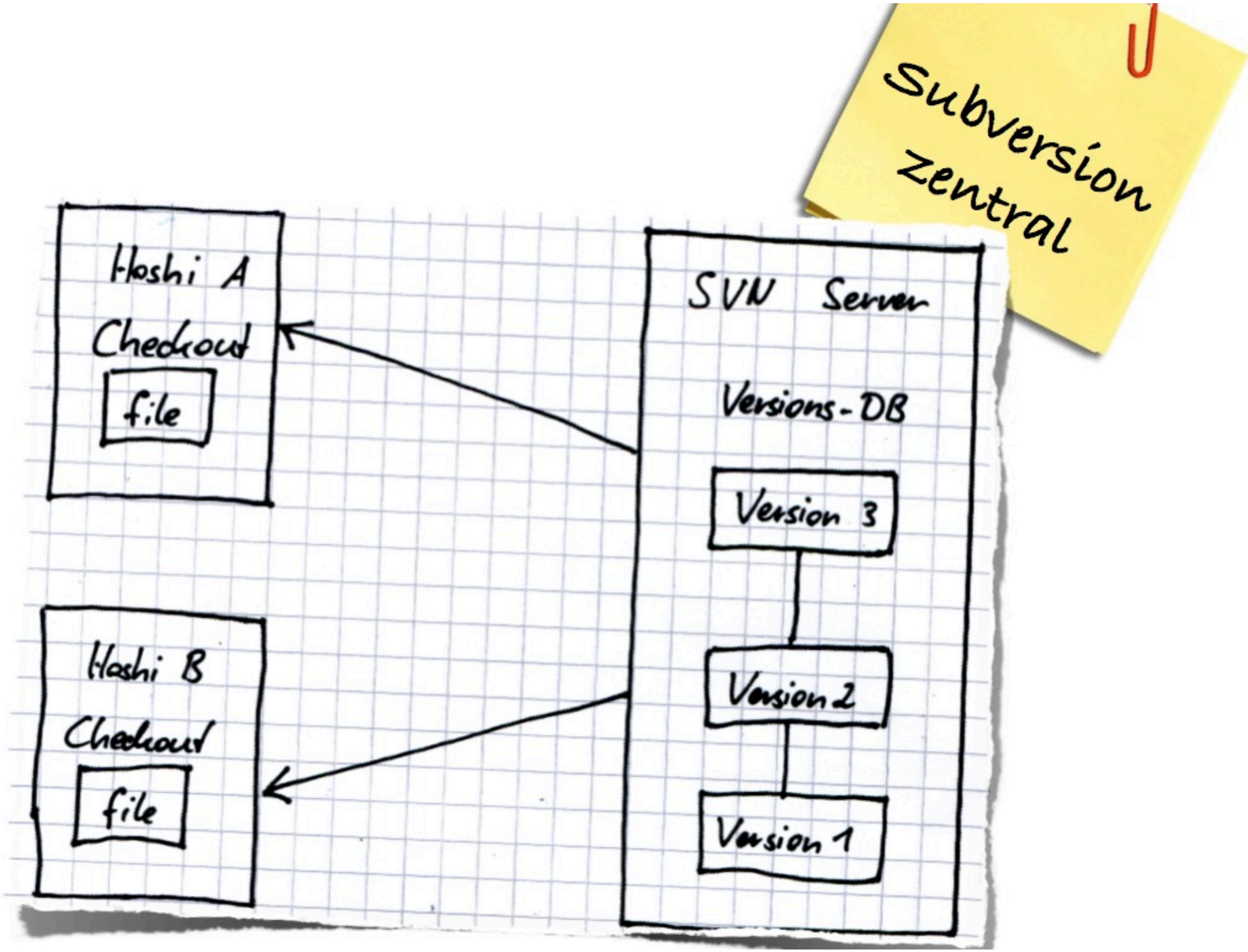
Protokollieren
Archivieren
Wiederherstellen

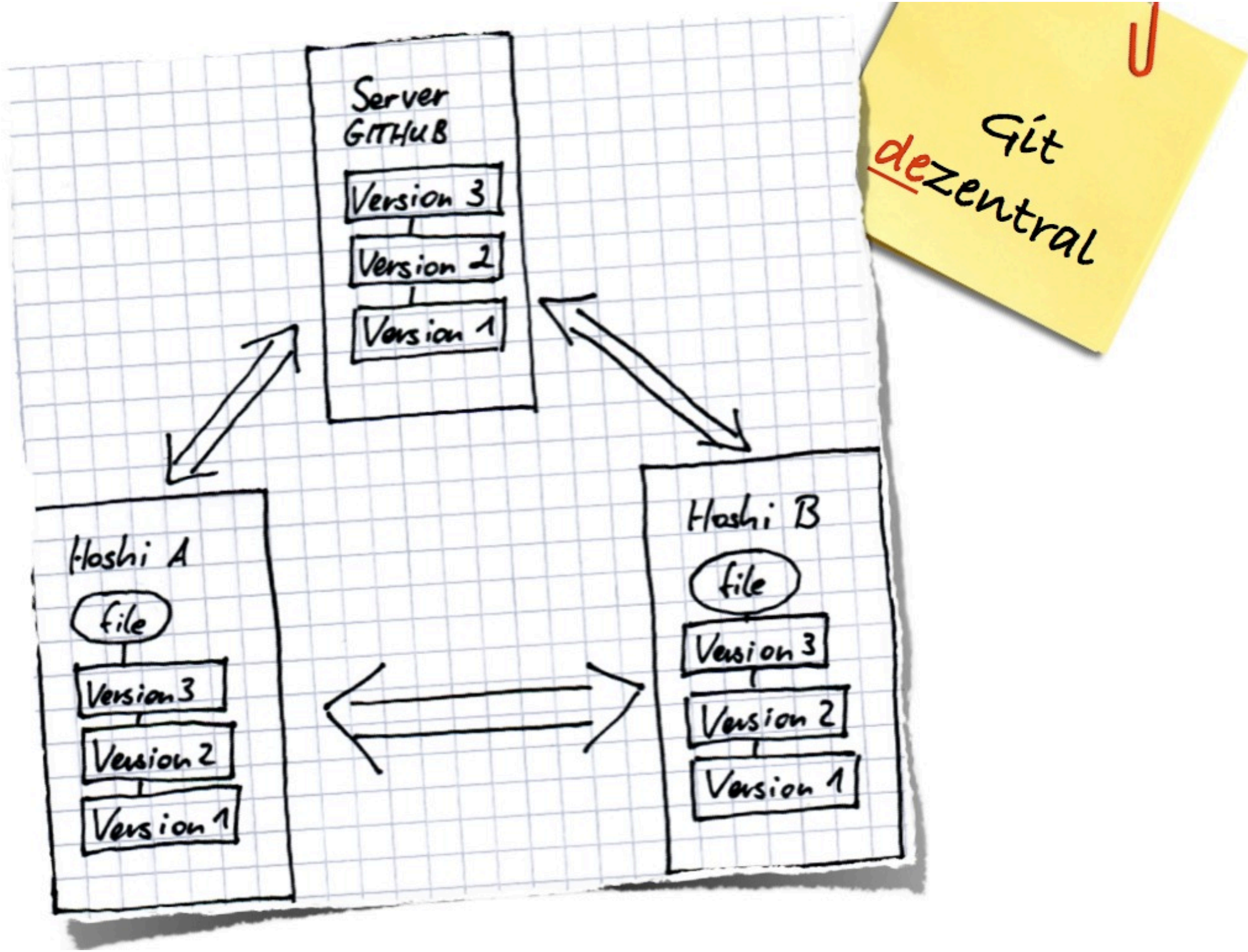



Come on in...
THE WATER'S FINE!
US NAVY
Klingt gut.
Was nehm ich?
Photograph by Leonard McCombe



SVN vs. GIT


Subversion zentral
Hoshi A
Checkout
file
SVN Server
Versions-DB
Version 3
Version 2
Version 1
Hoshi B
Checkout
file


Server
GITHUB
Version 3
Version 2
Version 1
Hoshi A
file
Version 3
Version 2
Version 1
Hoshi B
file
Version 3
Version 2
Version 1
Git
dezentral

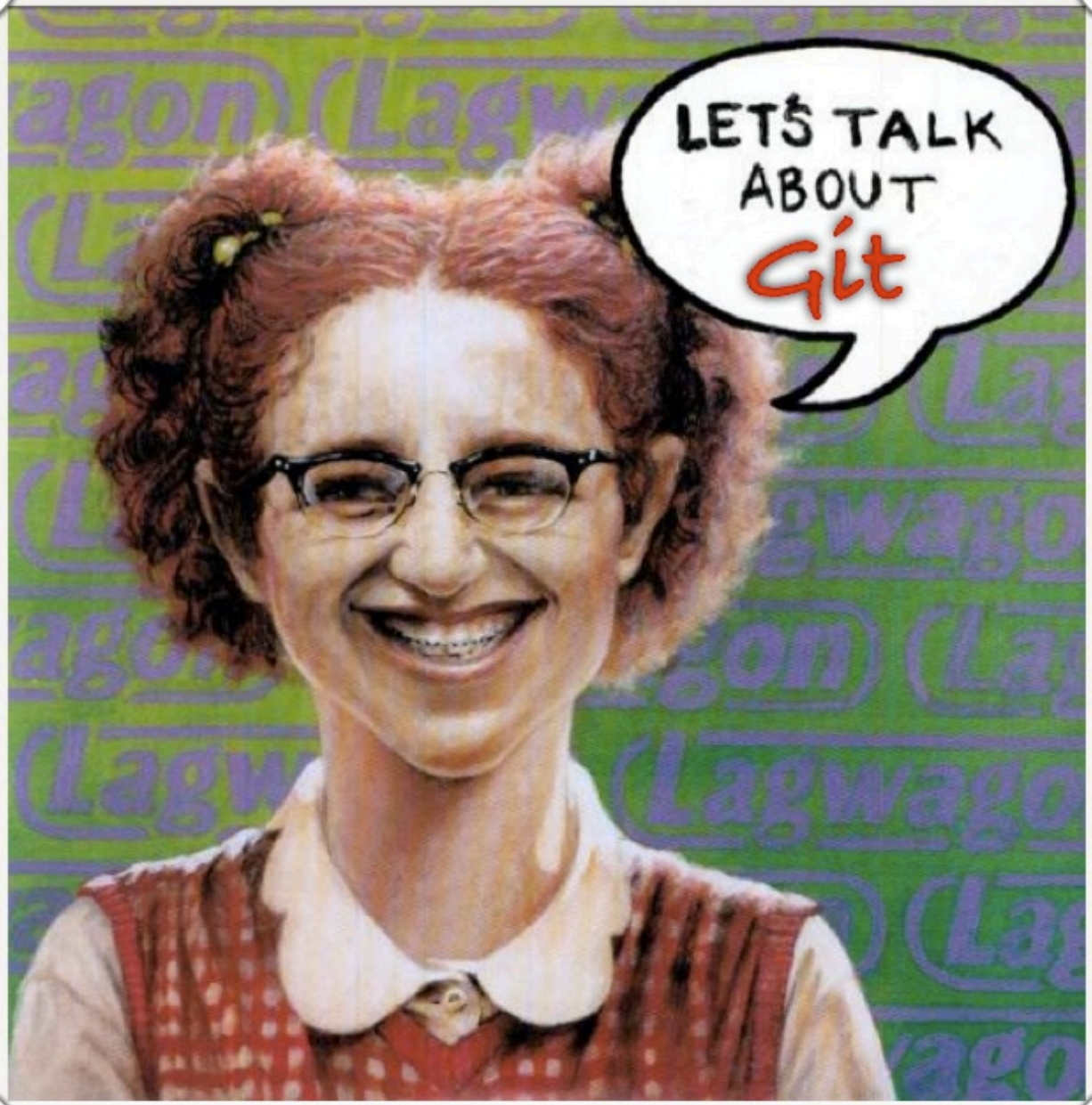

HMMM...
ICH GLAUB'
ICH NEHM'
DIE BLONDE.
? !

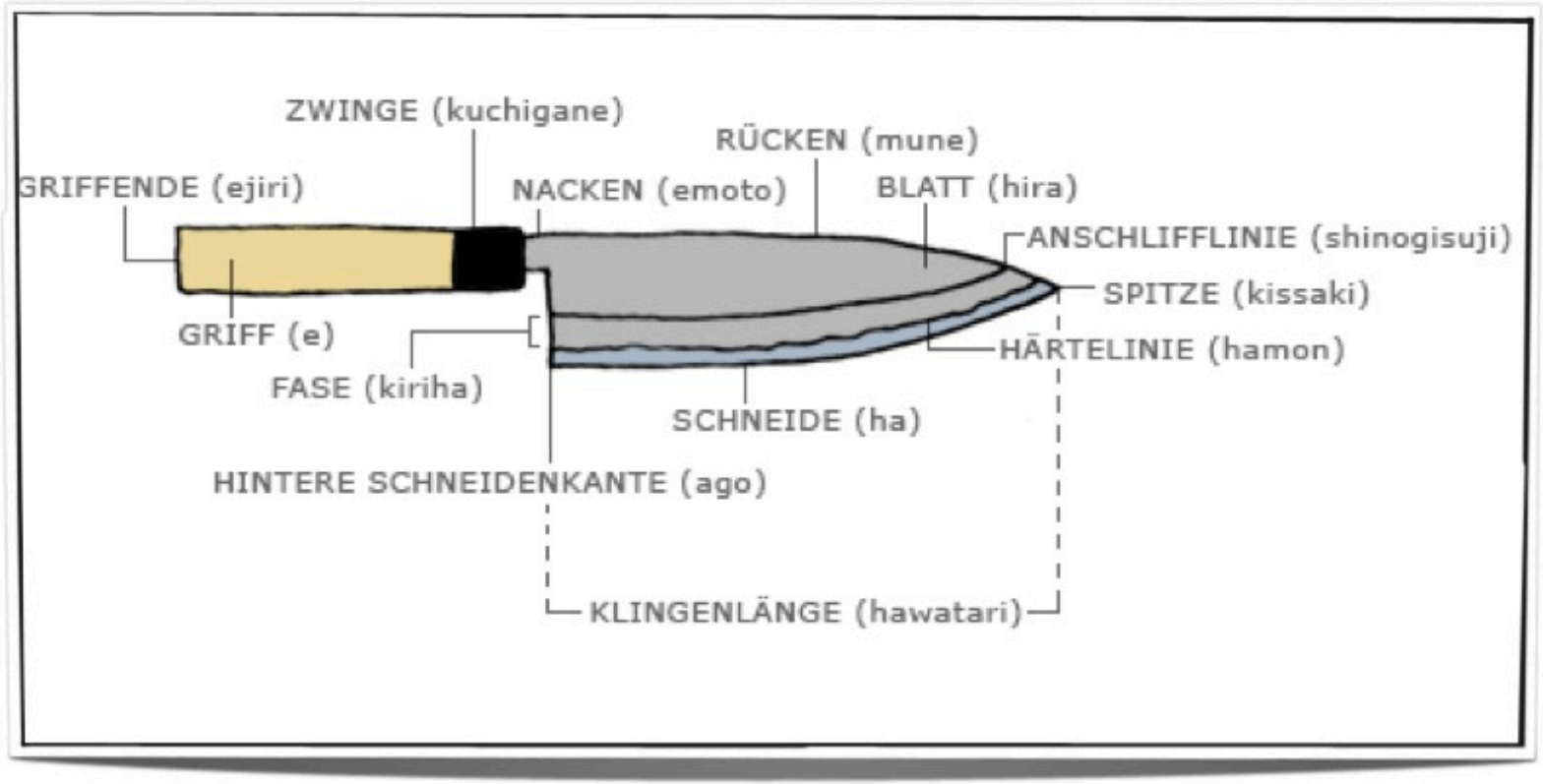
* Wird sehr gut von anderen Tools unterstützt
* Lässt sich "relativ" leicht verstehen
* Ihr könnt zentrales Rechtemanagement betreiben

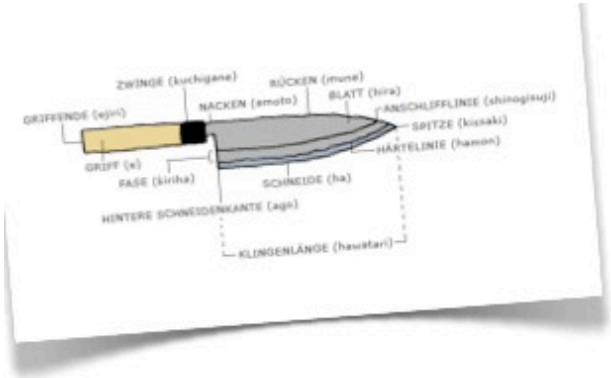
* Langsam
* Historie ist dumm
* Branching ist ziemlich anstrengend
* Ohne Server unbrauchbar

* Sau schnell
* Merge ist einfach
* Branch, Branch, Branch ...
* Intelligente Historie
* Geht auch ohne Server

* Steiniger Einstieg
* ~~Think dif~~ <!-- © violation here! -->
  Paradigmenwechsel
* Linux/Unix simpel, <omg>Windows</omg>
* Rechtemanagement


LETS TALK
ABOUT
Git


ZWINGE (kuchigane)
RÜCKEN (mune)
GRIFFENDE (ejiri)
NACKEN (emoto)
BLATT (hira)
ANSCHLIFFLINIE (shinogisuji)
SPITZE (kissaki)
GRIFF (e)
HÄRTELINIE (hamon)
FASE (kiriha)
SCHNEIDE (ha)
HINTERE SCHNEIDENKANTE (ago)
KLINGENLÄNGE (hawatari)



Begriffe

heads

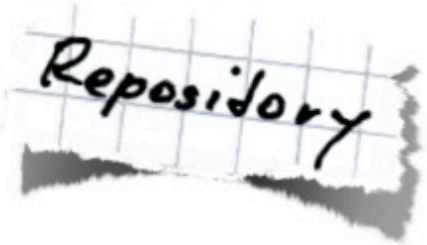
Branch

Merging

Rebase

Repository

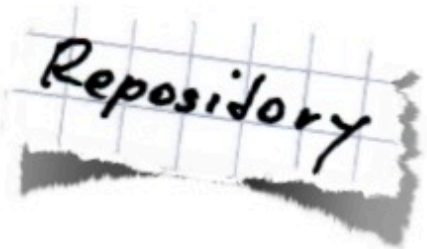
Commit Objects



* Git verwaltet eine Sammlung von Dateien und deren Änderungen über die Zeit

* All diese Informationen werden in der Datenstruktur Repository gespeichert

```
indigo@blinky:~/playground/myGitProject$ l
total 0
drwxr-xr-x   3 indigo  staff   102B Nov 10 14:35 .
drwxr-xr-x   3 indigo  staff   102B Nov 10 14:35 ..
drwxr-xr-x  10 indigo  staff   340B Nov 10 14:35 .git
indigo@blinky:~/playground/myGitProject$
```



Wie kommt man da hin?

* Ihr braucht git
* Nen bissle Grundkonfiguration

```
git config --global user.name "FirstName LastName"
git config --global user.email "user@example.com"

git config --global color.branch "auto"
git config --global color.status "auto"
git config --global color.diff "auto"

cd PATH/TO/MY/AWESOME/PROJECT
git init
```

Commit Objects

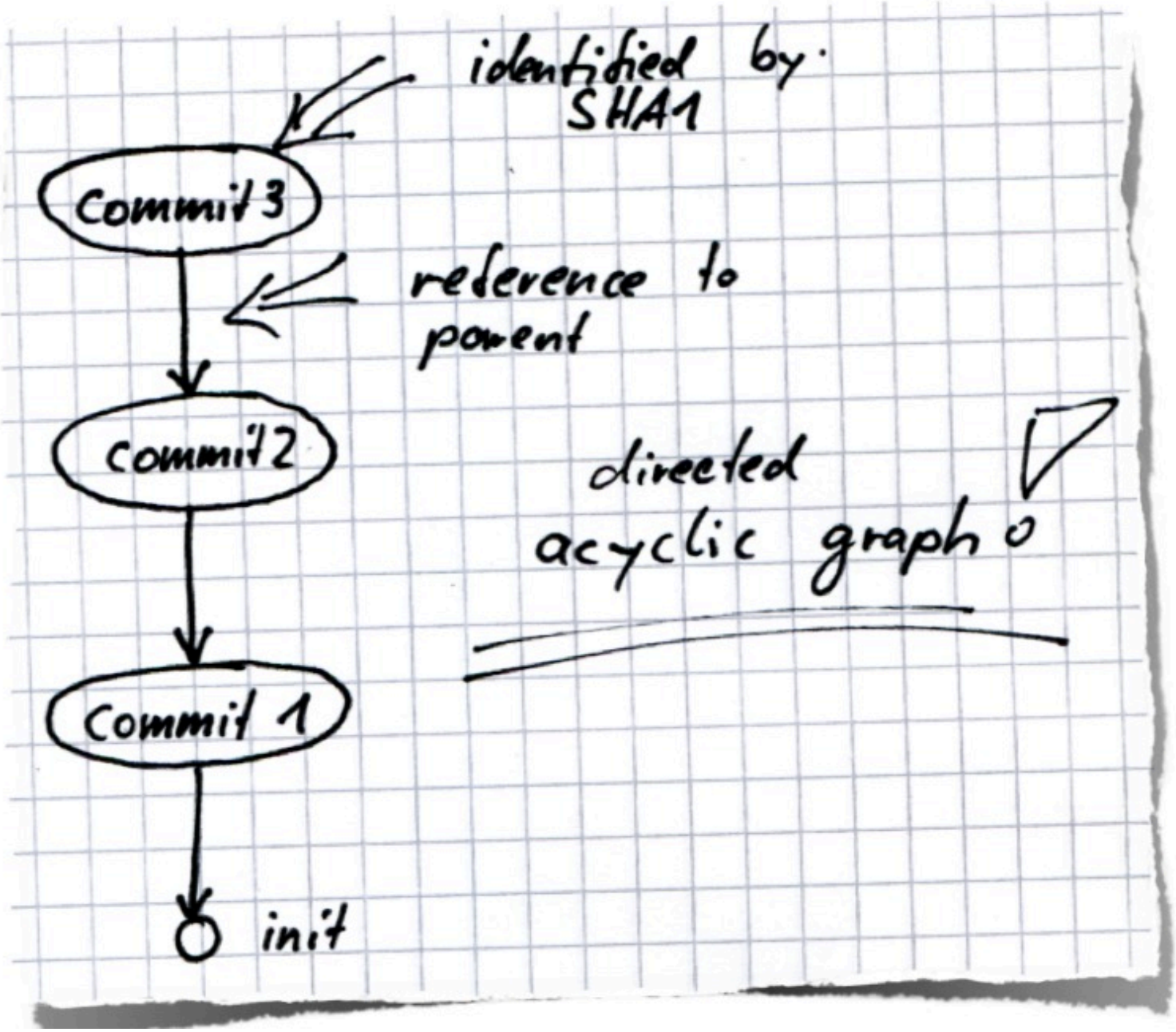
* Eine Sammlung von Dateien, die einen Projektstand wiederspiegeln

* Die Referenz auf sein parent commit object!

* SHA1 name, 40 Zeichen identifizieren das commit object. Erzeugter Hash von relevanten Teilen des commits -> identische commits haben den gleichen Namen

Was bedeutet das?

Commit Objects



DIE git IDEE?!

Versionskontrolle ist die Manipulation des Graphen mit all seinen commits.

Wenn ihr Abfragen oder Änderungen macht, kann es euch helfen das im Hinterkopf zu behalten.

* Ein **head** ist eine Referenz auf ein commit object

* By default gibts in jedem Repository einen head der **master** genannt wird

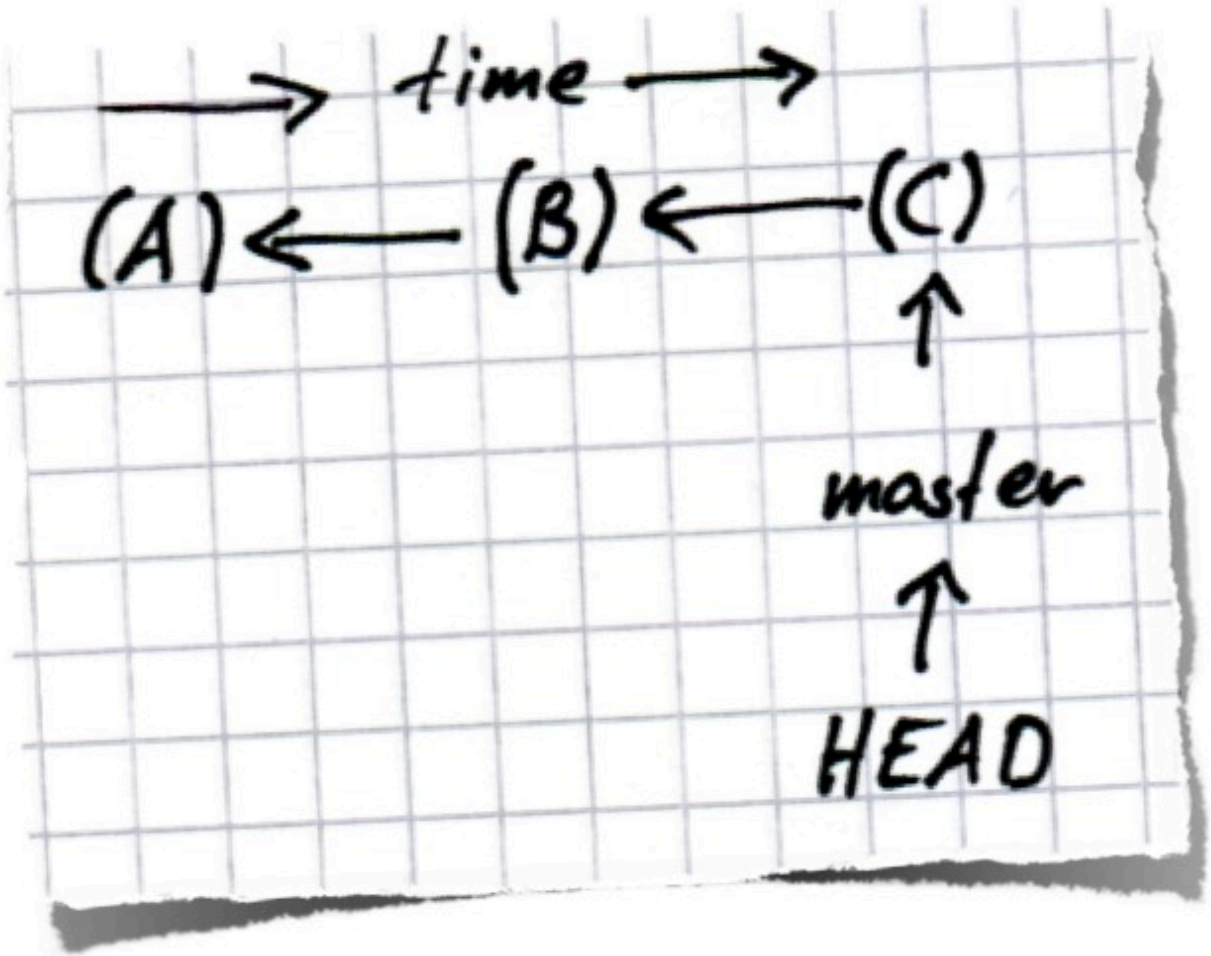
* Ein Repository kann beliebig **viele heads** haben

* Der aktuell aktive head wird **HEAD** genannt

Ok wir haben was zum spielen!

```
git init
git commit
git log
git status
git diff
git mv / git rm
```

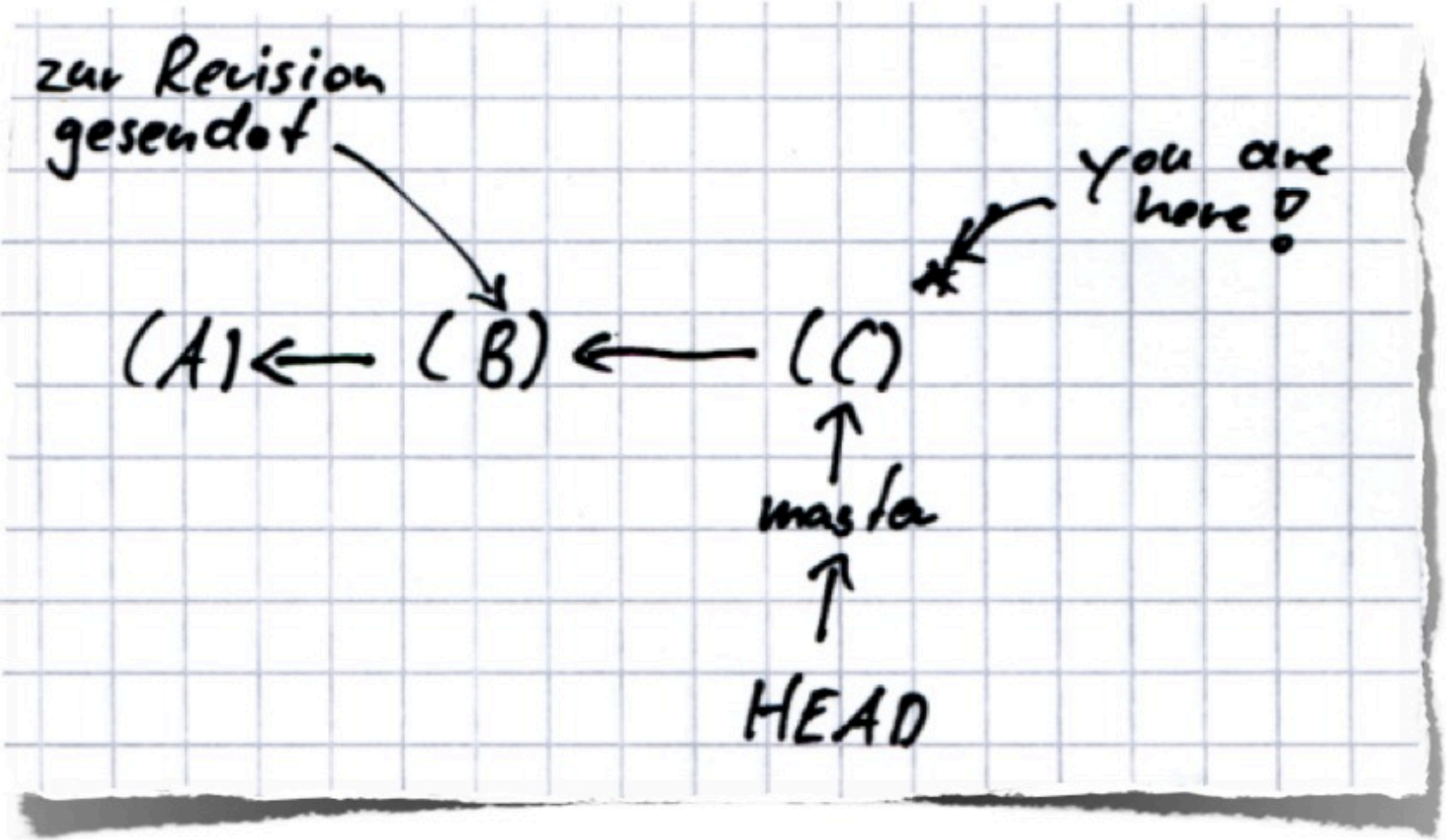
Ok wir haben was zum spielen!



Ein **branch** ist eine Referenz
auf ein commit object

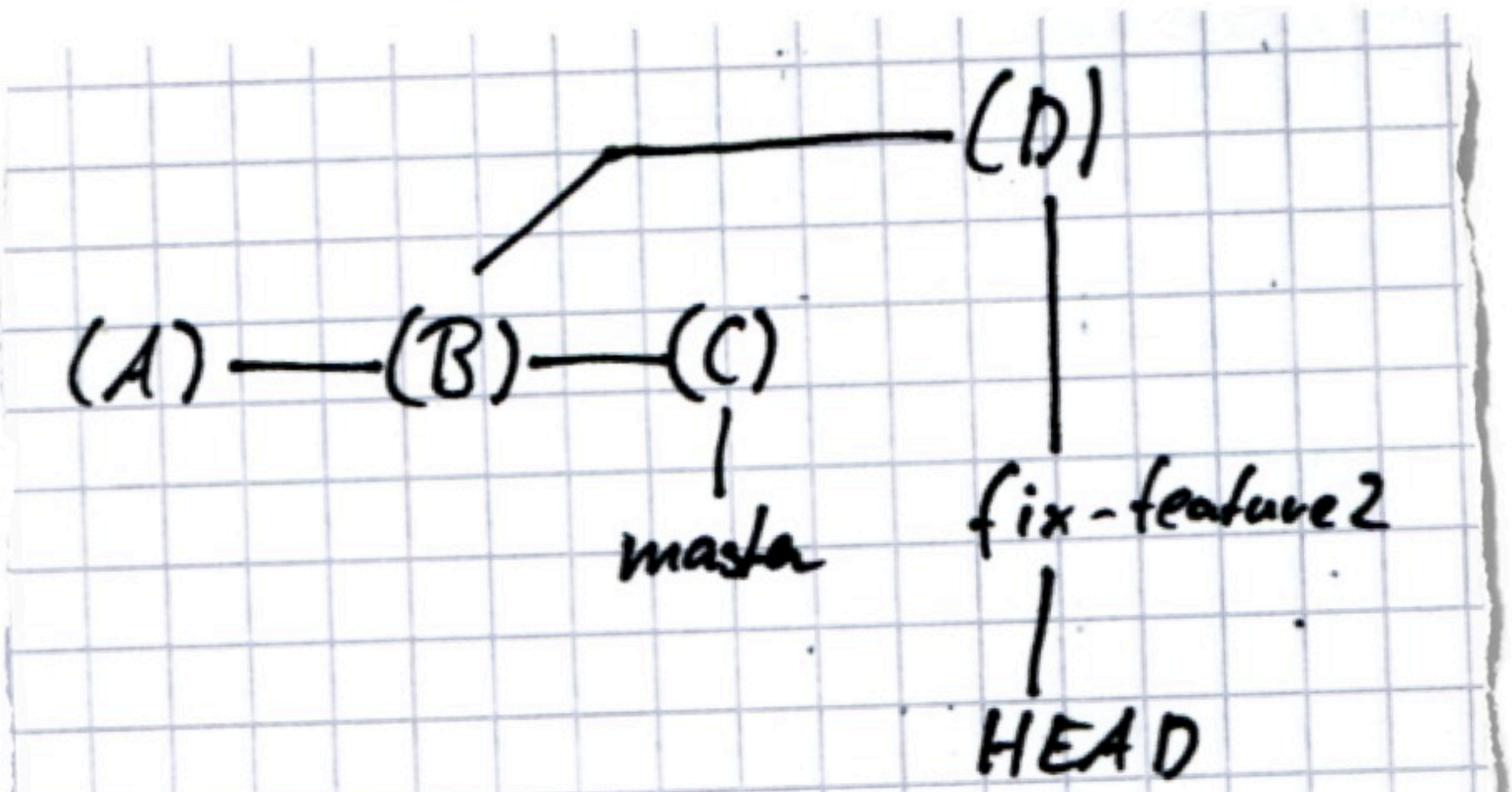
Branch

Beispiel:




```
git branch
git branch [new-head-name] [reference-to-(B)]
git checkout
```

Branch



* Branches zum implementieren neuer Features

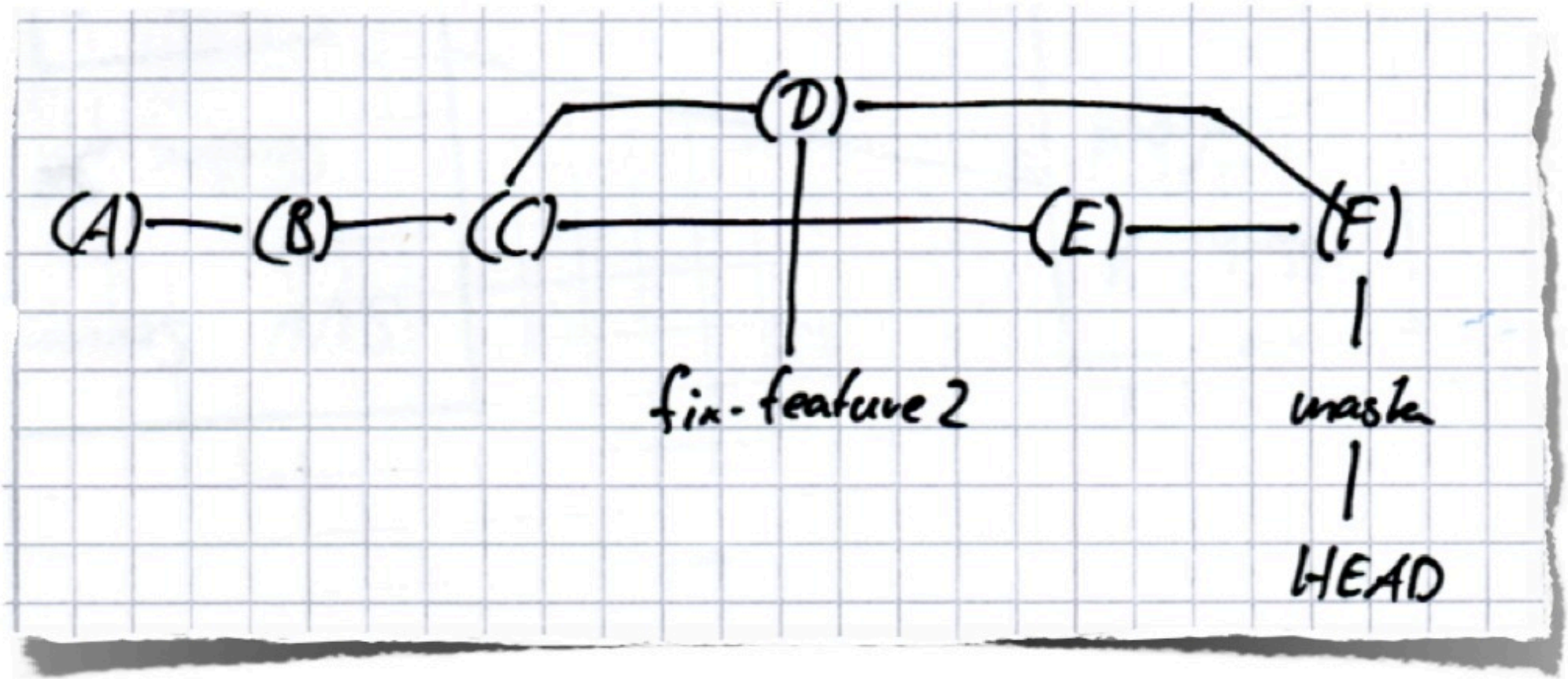
* Branches beinhalten "halbfertiges"

* aus master wird released (main/trunk)

* Jeder entwickler branched, commits können IMMER gemacht werden, egal ob fertig oder nicht

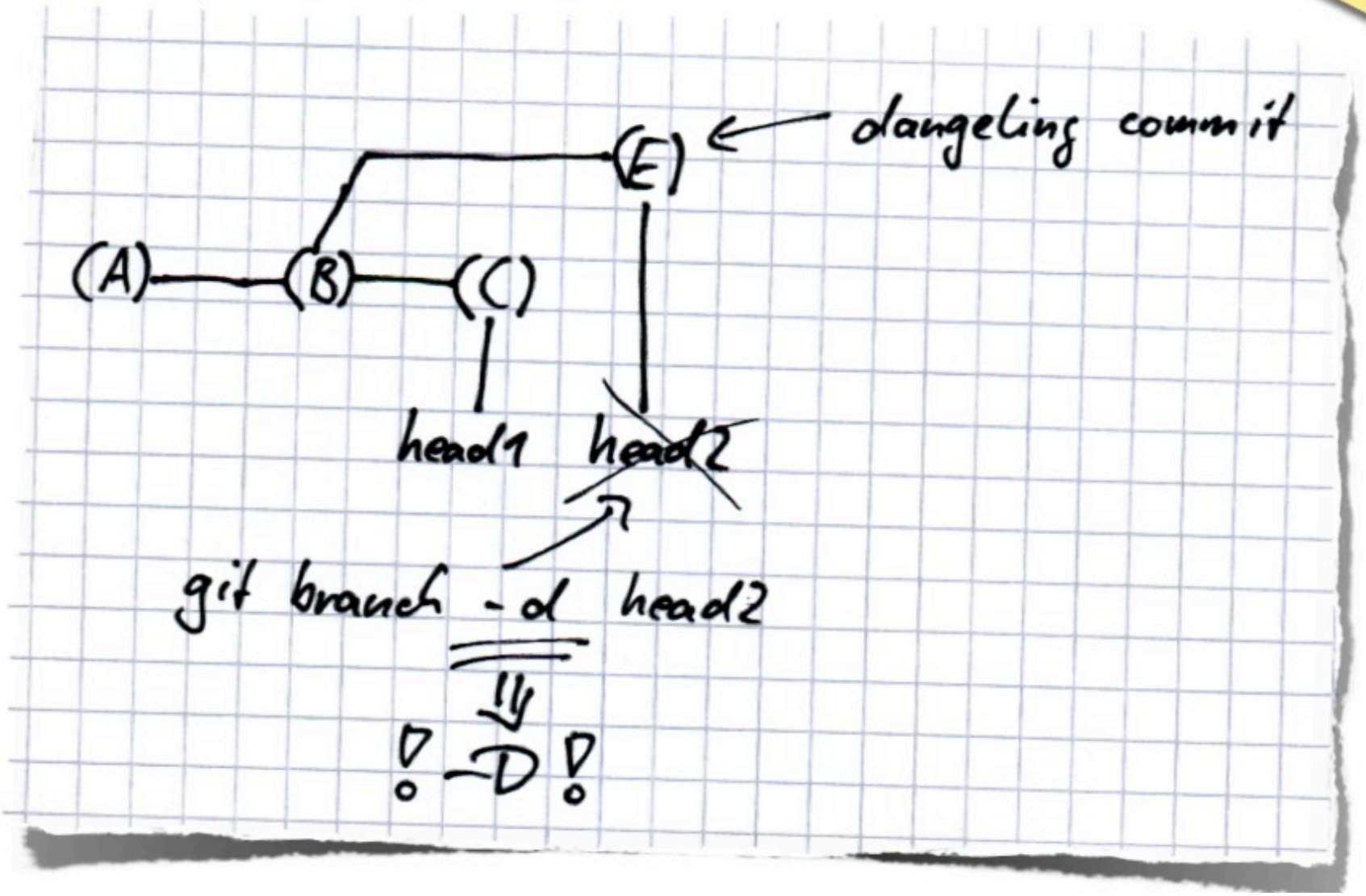
* Commits sind billig, es gibt KEINEN Grund nicht zu commiten!

Merging



deleting a branch

```
git branch -d [head]
```

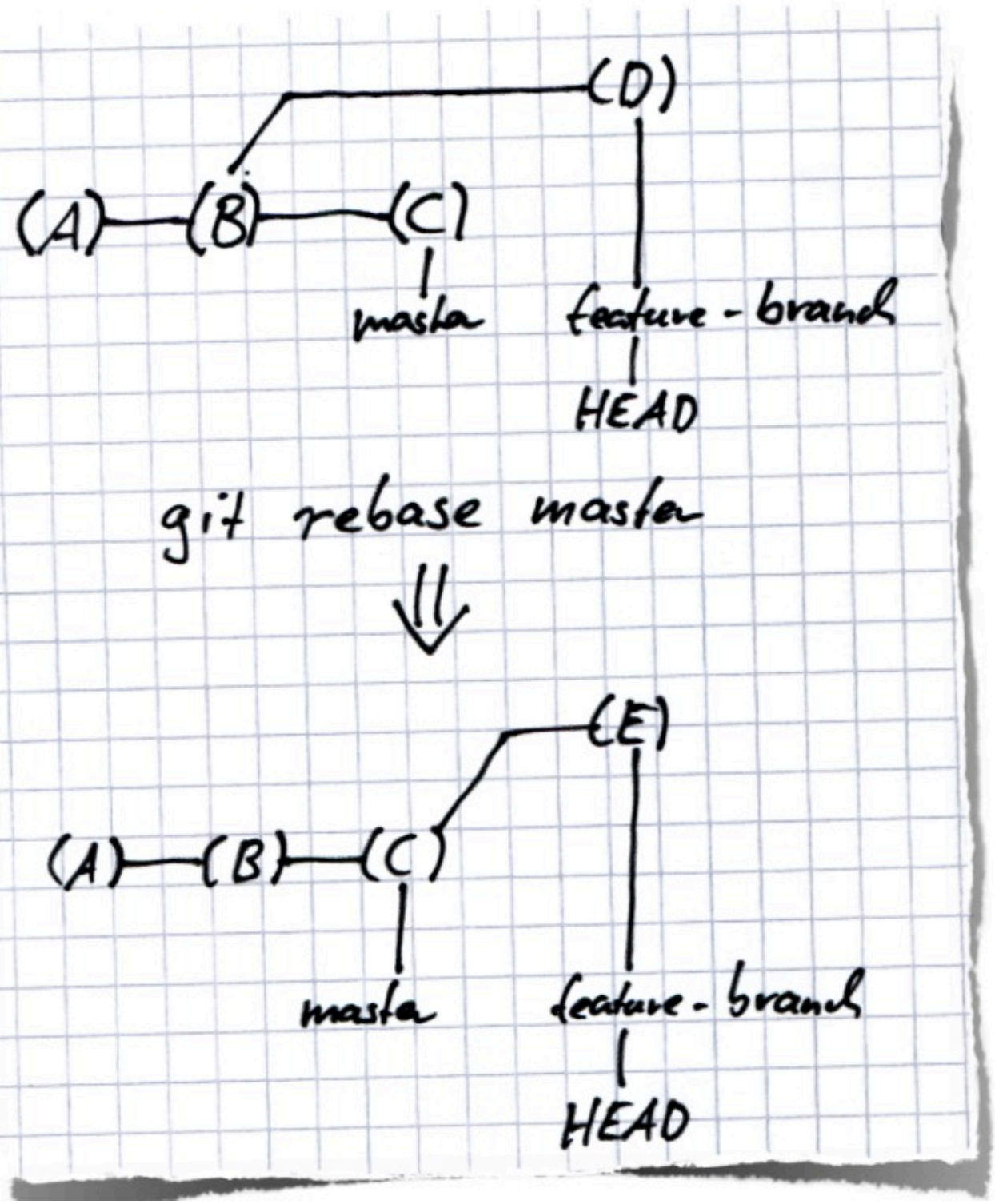


merge pattern

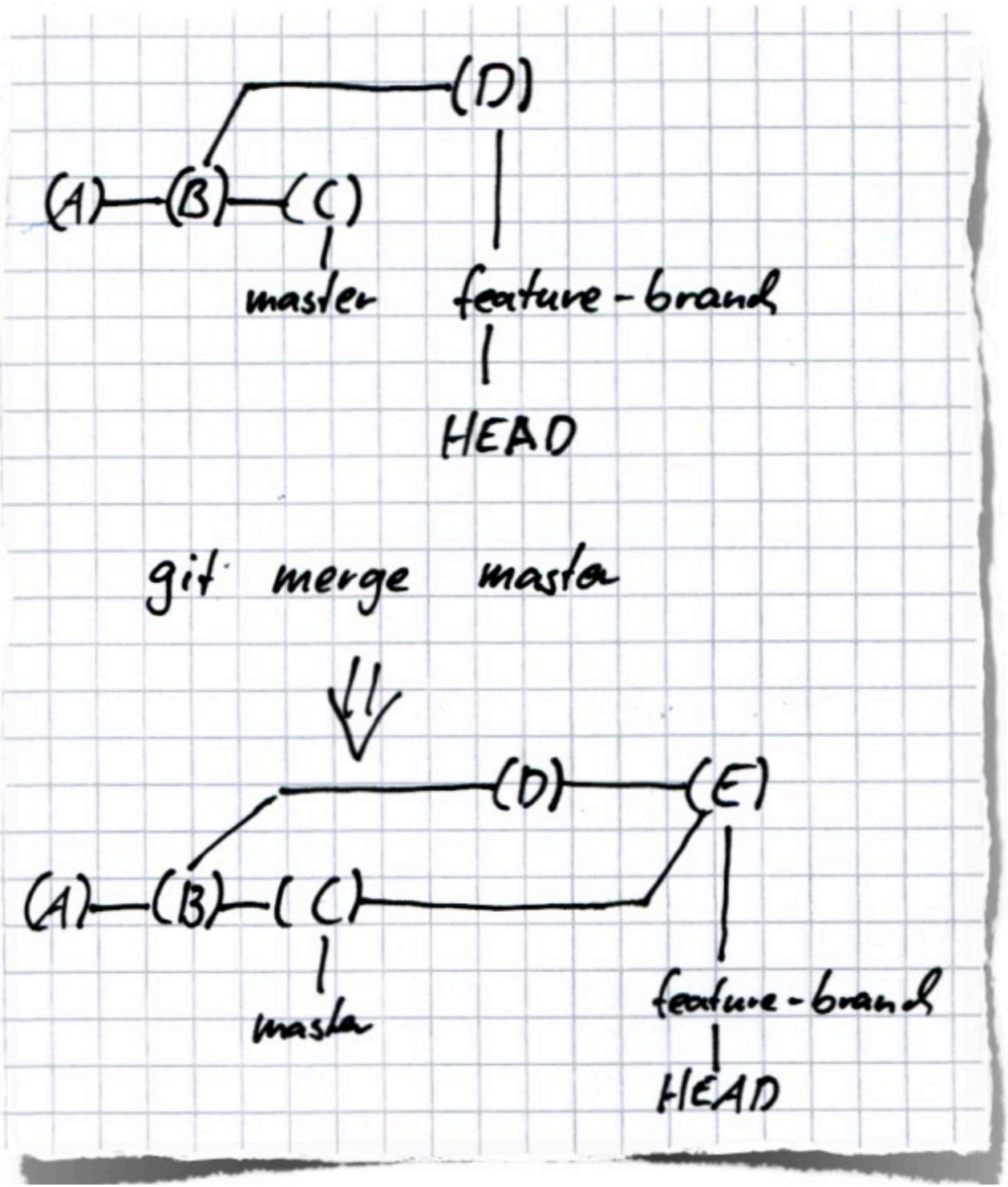
Merge hat meistens zwei Gründe:

* Features vom branch in den master zum release ziehen
* Bugfixes und features aus dem master in euren feature-branch ziehen um commit Konflikte zu reduzieren und Bugs gefixt zu bekommen
* Nachteil von dem da oben: Eure feature branch hat nen haufen merge commits
* Hier kann rebasing ins Spiel kommen. Hat aber auch keinen Kuchen :)

Rebase


(D)
(A)
(B)
(C)
master
feature - branch
HEAD
git rebase master
(E)
(A)
(B)
(C)
master
feature - branch
HEAD

Merging


(D)
(A)
(B)
(C)
master
feature-brand
HEAD
git merge master
(D)
(E)
(A)
(B)
(C)
master
feature-brand
HEAD


```
git reset --hard HEAD
git clean -fdx
git format-patch -M -C [head]
```

* SmartGit - Non-Plus-Ultra

* gitk

* Charles Duan, Understanding Git Conceptually

* Versionskontrolle mit Git von Jon Loeliger aus dem Verlag O'Reilly