Git im praktischen Einsatz

Vladimir Dobriakov

gearconf 2012, Düsseldorf

www.mobile-web-consulting.de

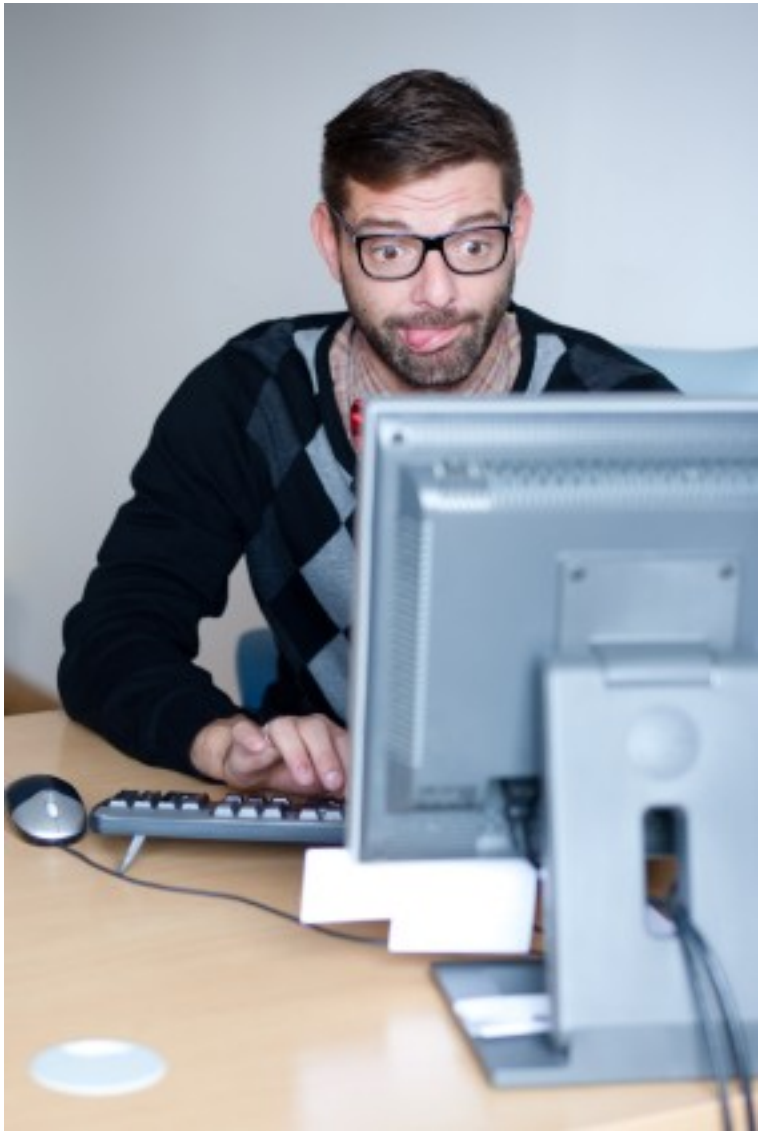

93 000 links
on delicious.com
2 Mio.
Repositories
on github.com
articles
blog
branching
deployment
framework
git
github
google
help
hosting
howto
html
html5
javascript
js
library
mac
master
programming
rails
server
software
ssh
subversion
twitter
version
web
webdesign
webdev
wiki
workflow
www

git ‹command›

```
add
add--interactive
am
annotate
apply
archive
bisect
bisect--helper
blame
branch
bundle
cat-file
check-attr
check-ref-format
checkout
checkout-index
cherry
cherry-pick
clean
clone
commit
commit-tree
config
count-objects
credential-cache
credential-cache--daemon
credential-store
daemon
describe
diff
diff-files
diff-index
diff-tree
difftool
difftool--helper
fast-export
fast-import
fetch
fetch-pack
filter-branch
fmt-merge-msg
for-each-ref
format-patch
fsck
fsck-objects
gc
get-tar-commit-id
grep
hash-object
help
http-backend
http-fetch
http-push
imap-send
index-pack
init
init-db
instaweb
log
lost-found
ls-files
ls-remote
ls-tree
mailinfo
mailsplit
merge
merge-base
merge-file
merge-index
merge-octopus
merge-one-file
merge-ours
merge-recursive
merge-resolve
merge-subtree
merge-tree
mergetool
mktag
mktree
mv
name-rev
notes
pack-objects
pack-redundant
pack-refs
patch-id
peek-remote
prune
prune-packed
pull
push
quiltimport
read-tree
rebase
receive-pack
reflog
relink
remote
remote-ext
remote-fd
remote-ftp
remote-ftps
remote-http
remote-https
remote-testgit
repack
replace
repo-config
request-pull
rerere
reset
rev-list
rev-parse
revert
rm
send-pack
sh-i18n--envsubst
shell
shortlog
show
show-branch
show-index
show-ref
stage
stash
status
stripspace
submodule
symbolic-ref
tag
tar-tree
unpack-file
unpack-objects
update-index
update-ref
update-server-info
upload-archive
upload-pack
var
verify-pack
verify-tag
web--browse
whatchanged
write-tree
```

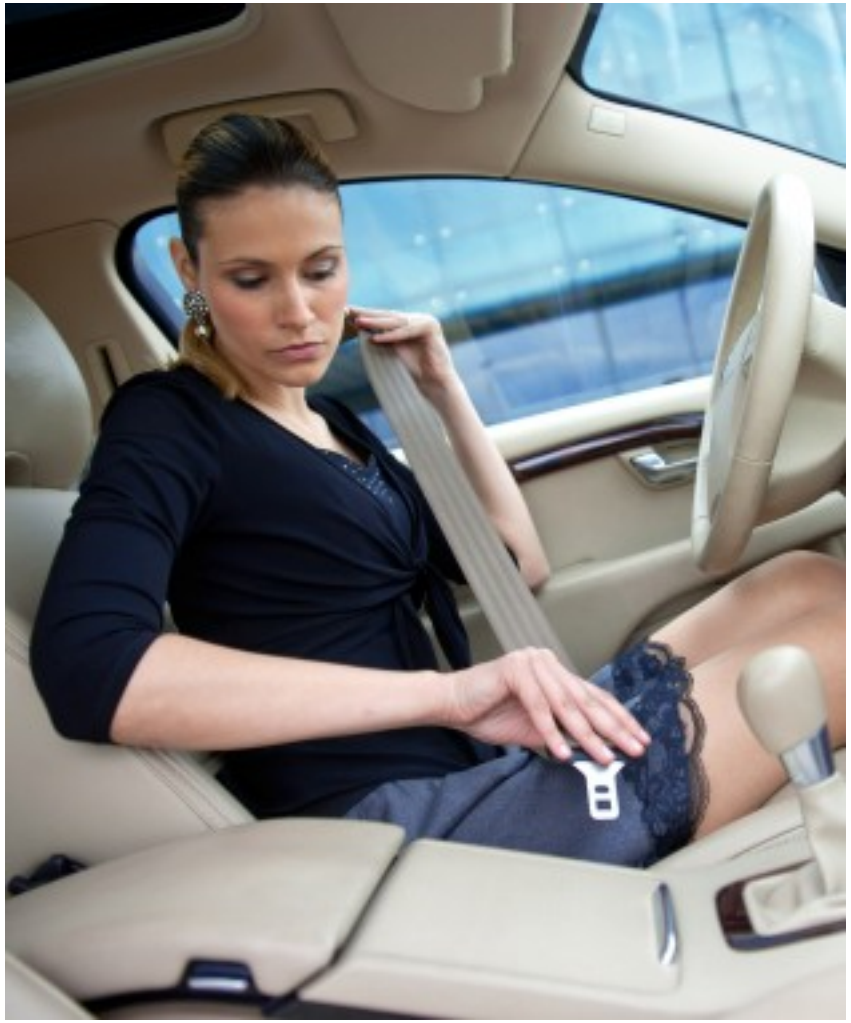
Einsatzszenarien

1. Kleines Projekt/ Einzelentwickler



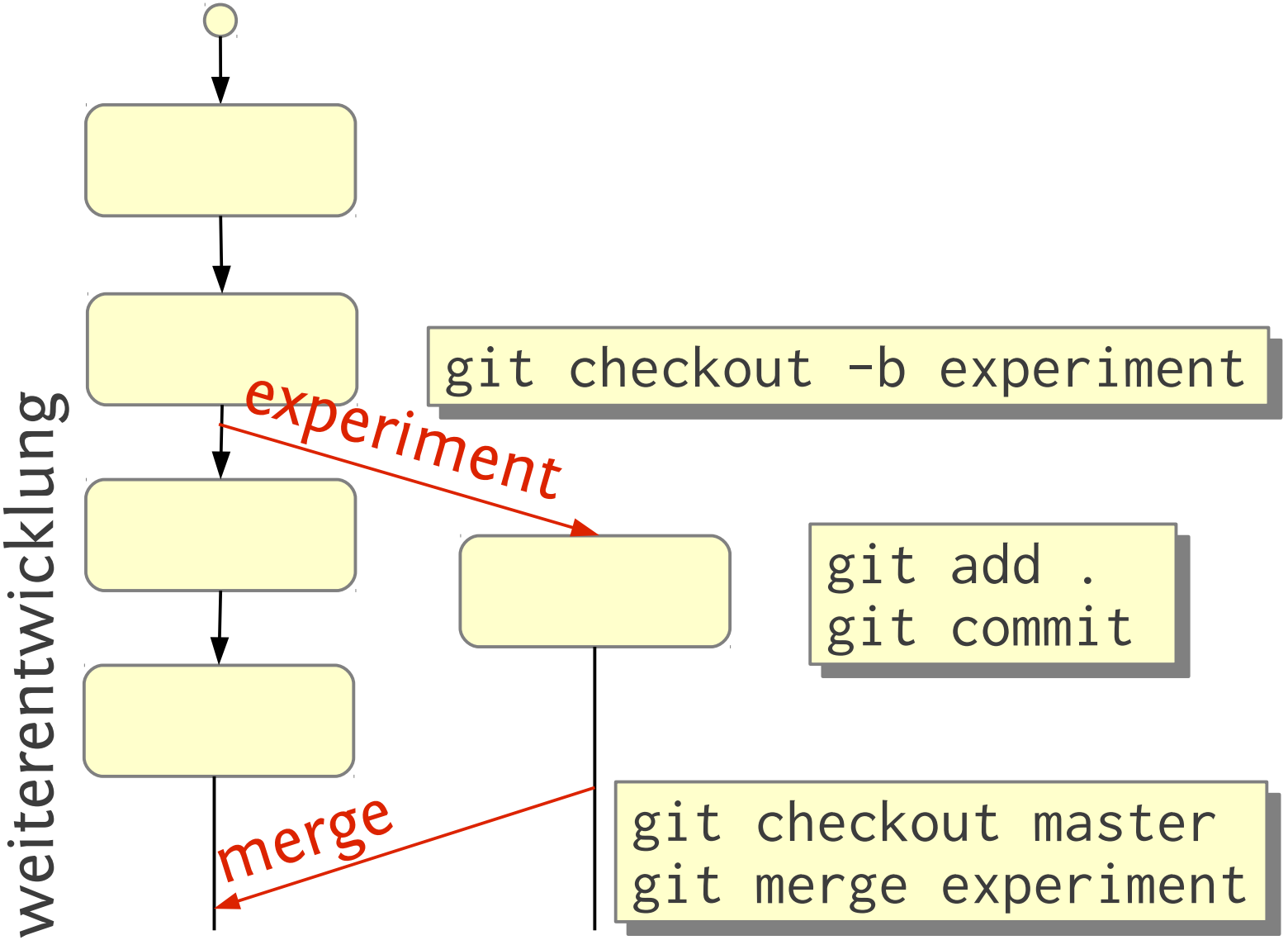
- Prototyp
- kleines Werkzeug
- Konfigurationsdateien

1. Ein sicheres Gefühl



```
git init
git add .
git commit -m \
        'Initial version'
# play around
git commit -am \
              'My changes'
# or throw away
git checkout .
```

1. Mainline + Experimente



1. Vorteile

* sofort, auch ohne Server nutzbar
* auch offline
* daher auch für Kleinstanwendungen geeignet

Szenario 2. OpenSource Projekt

Committers

vs.

Users

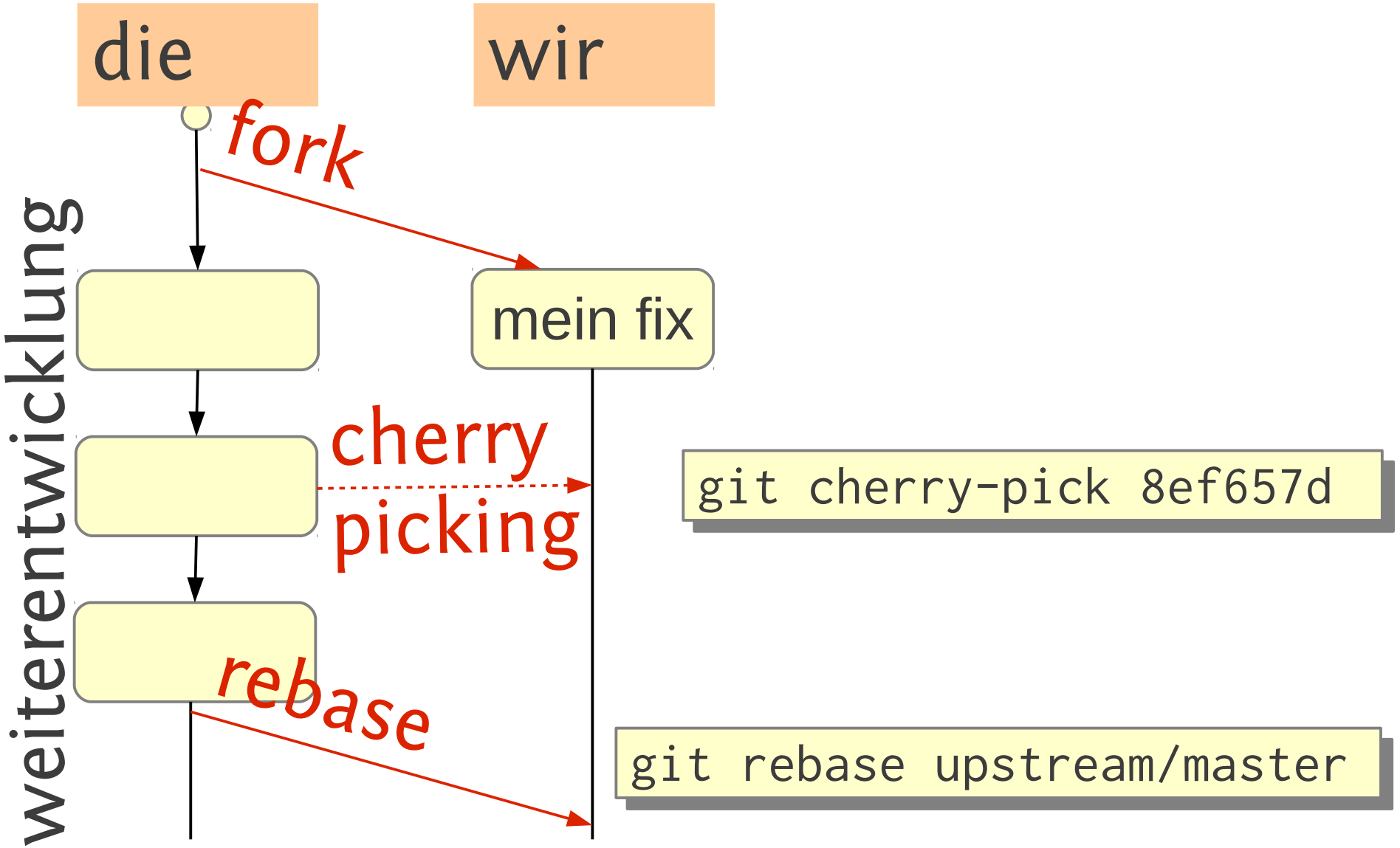
Szenario 2. OpenSource Projekt

Committers

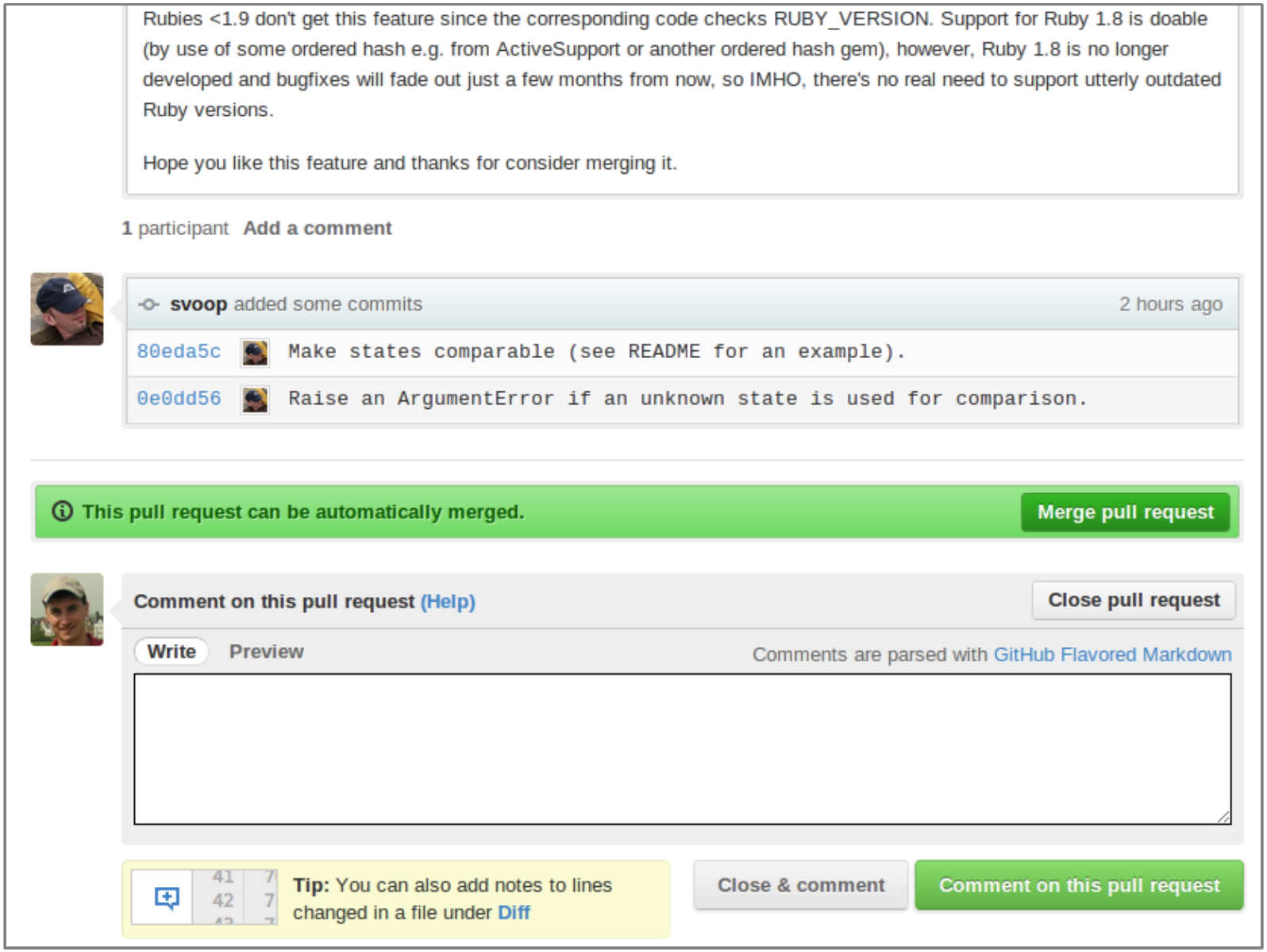
vs.

Losers

2. Fork you!


die
wir
fork
weiterentwicklung
mein fix
cherry
picking
git cherry-pick 8ef657d
rebase
git rebase upstream/master

2. Pull Request



2. Rasend schnell !

Rails

Linux Kernel

```
git clone https://github.com/rails/rails.git
```

300 000 Objekte
in 2 Minuten

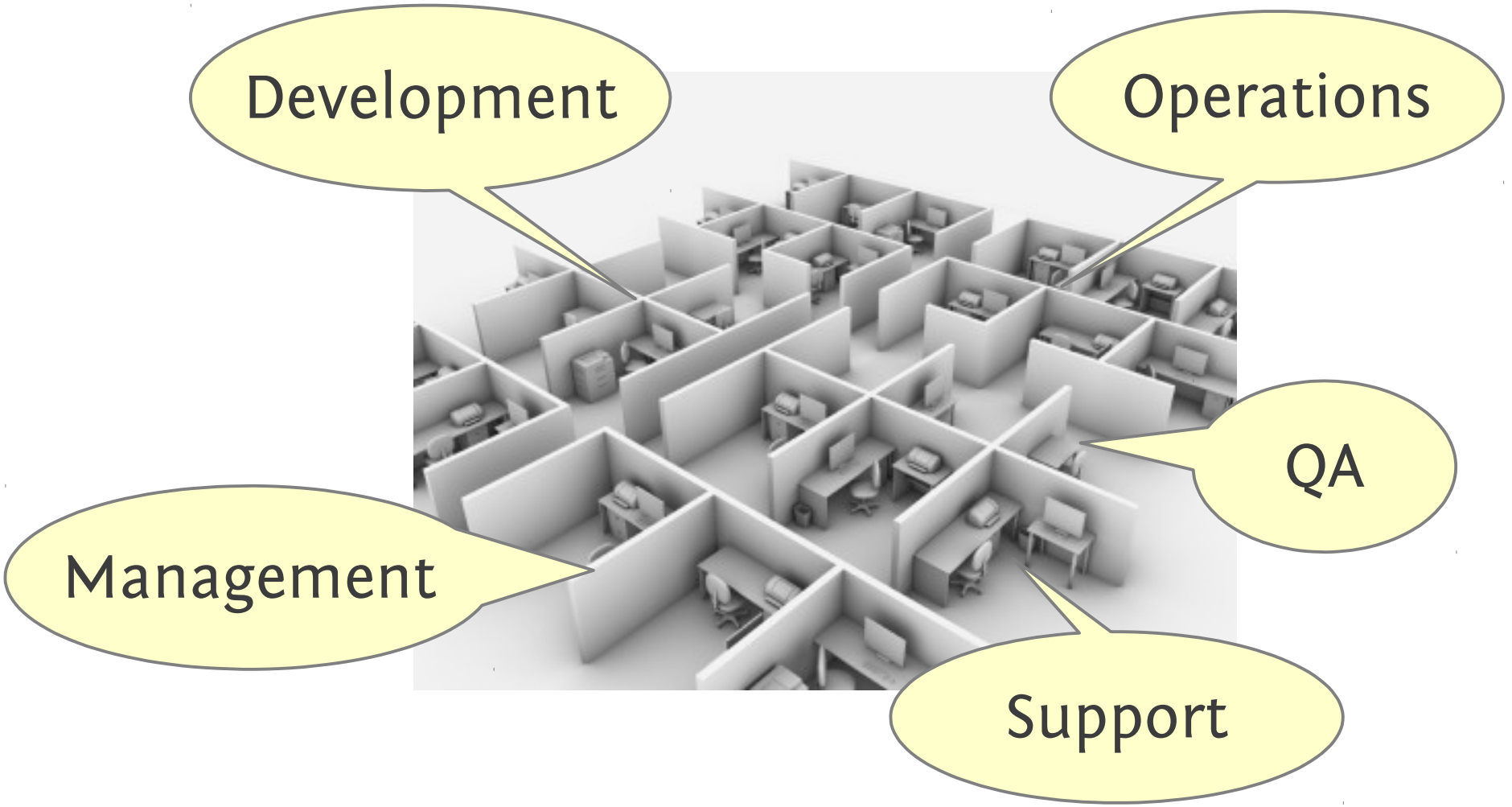
durchsuche
31 212 commits
in 185ms

```
git log --oneline validations/format.rb
```

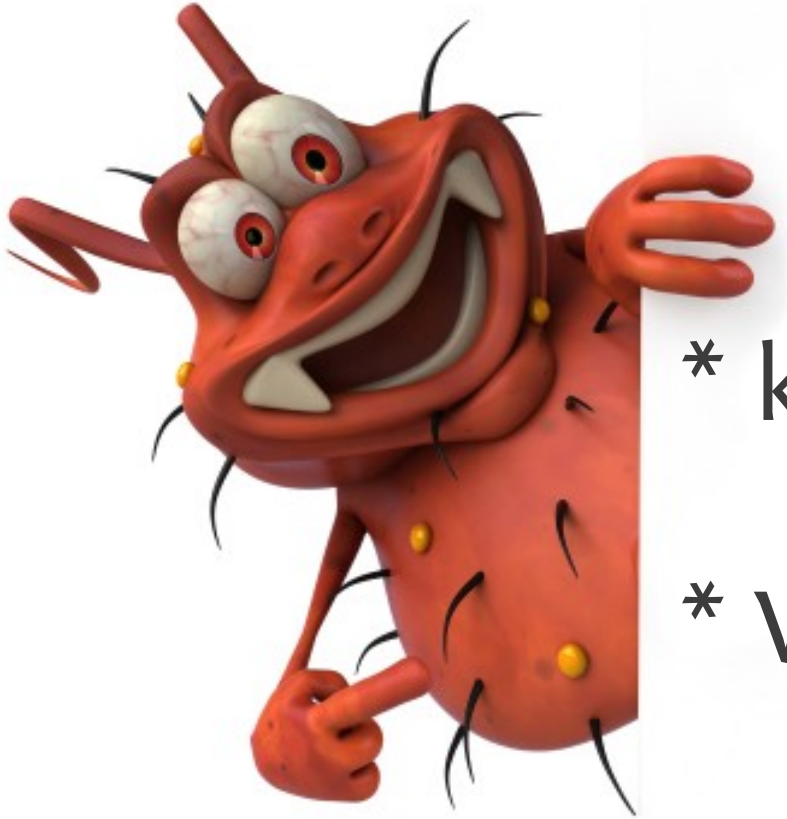
2. Vorteile

* eigene Fixes mit Weiterentwicklung kombinierbar
* ermöglicht reichhaltiges Ökosystem
* schlechter Maintainer wird von guten abgelöst
* schnelles Arbeiten auch bei großen Projekten

3. Enterprise-y



3. Bugs und Features in Großprojekten

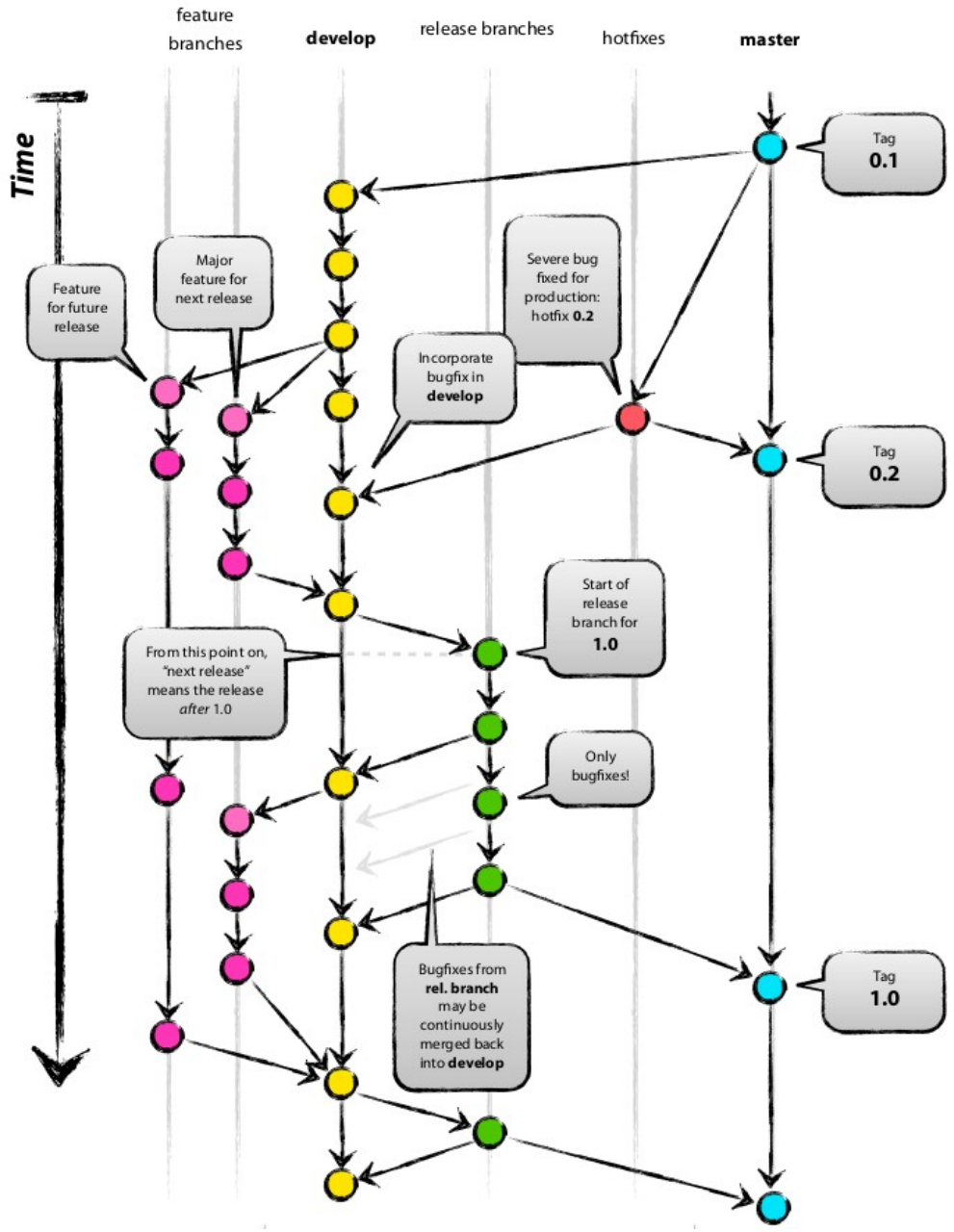


* kritisch (security) ? nicht

* Verschlimmbesserungen?

* langer Weg zu Produktion

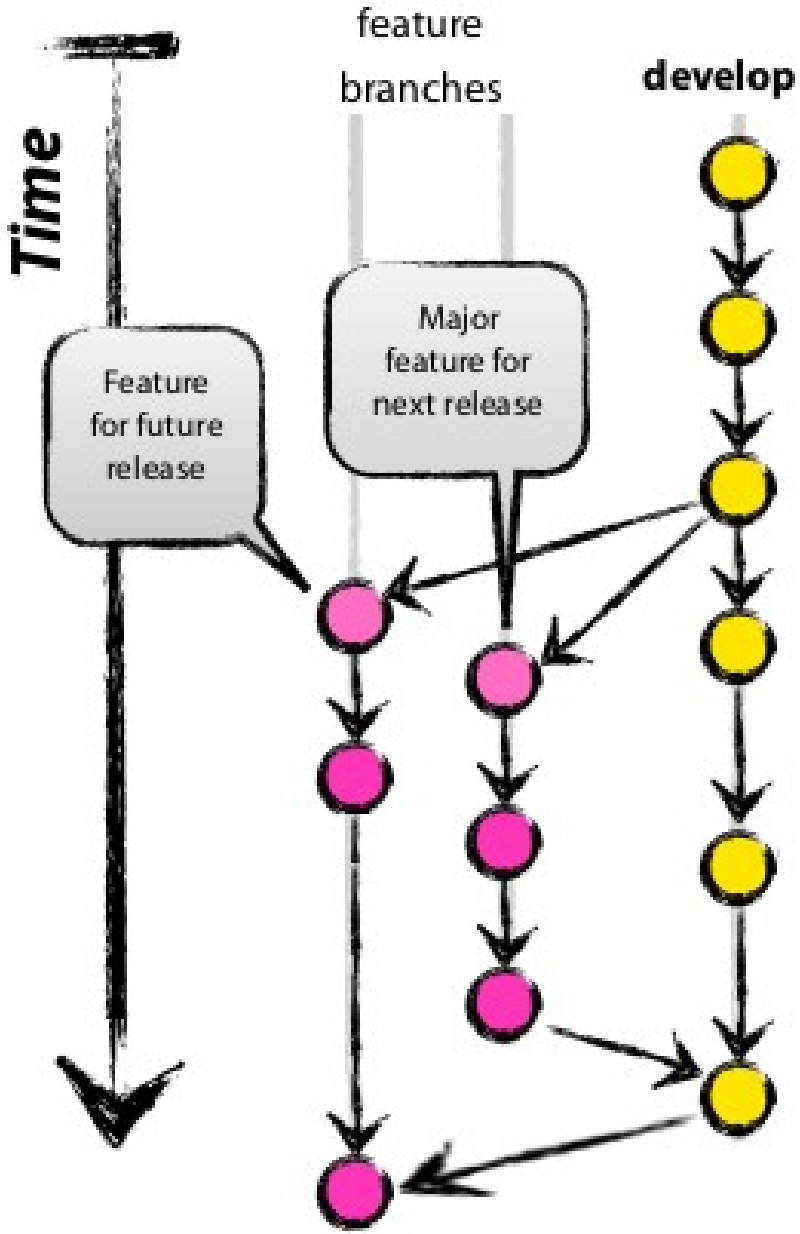
3. git-flow



+ Workflow

+ Werkzeuge

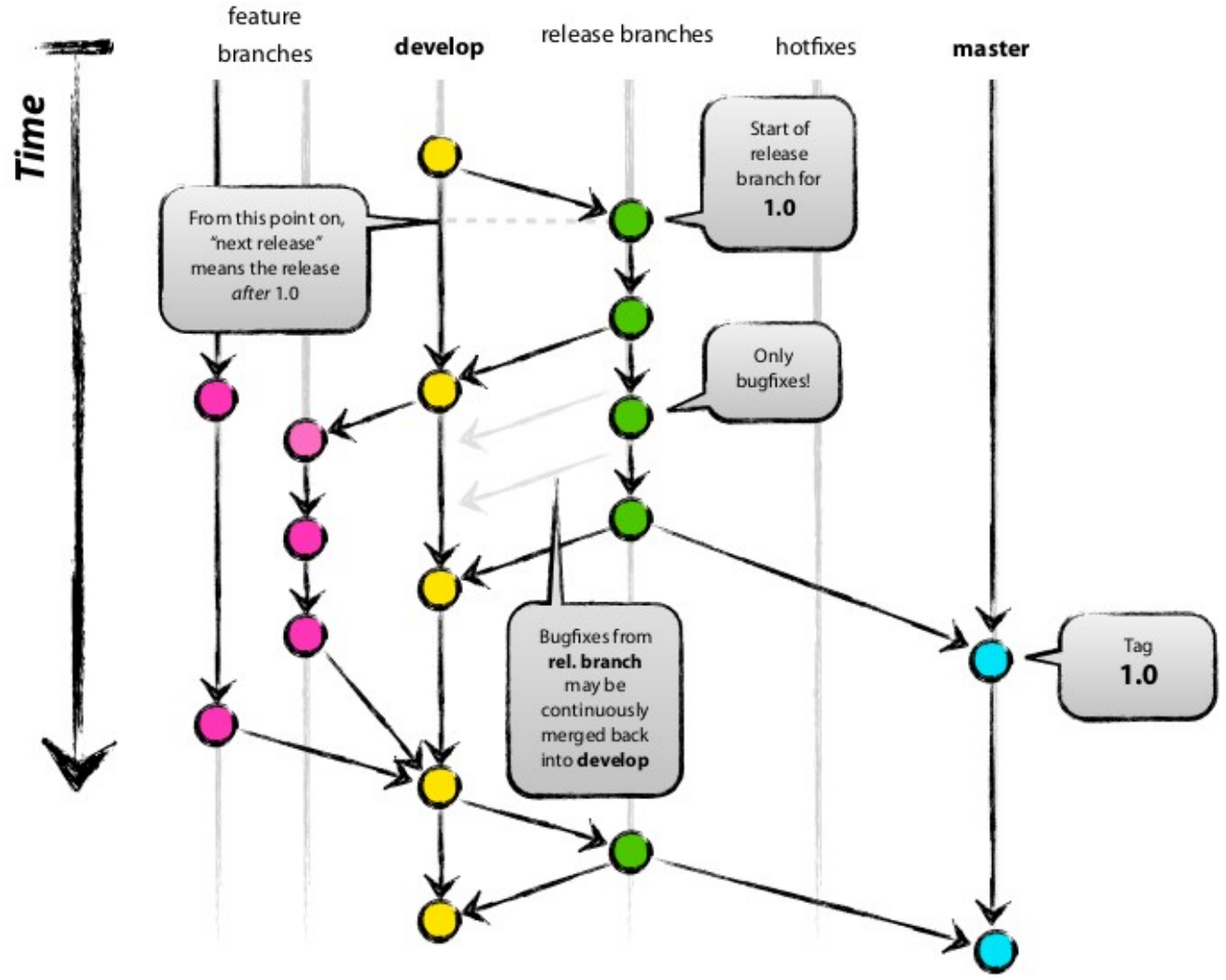
3. git-flow: Entwicklung




```
git flow init
```

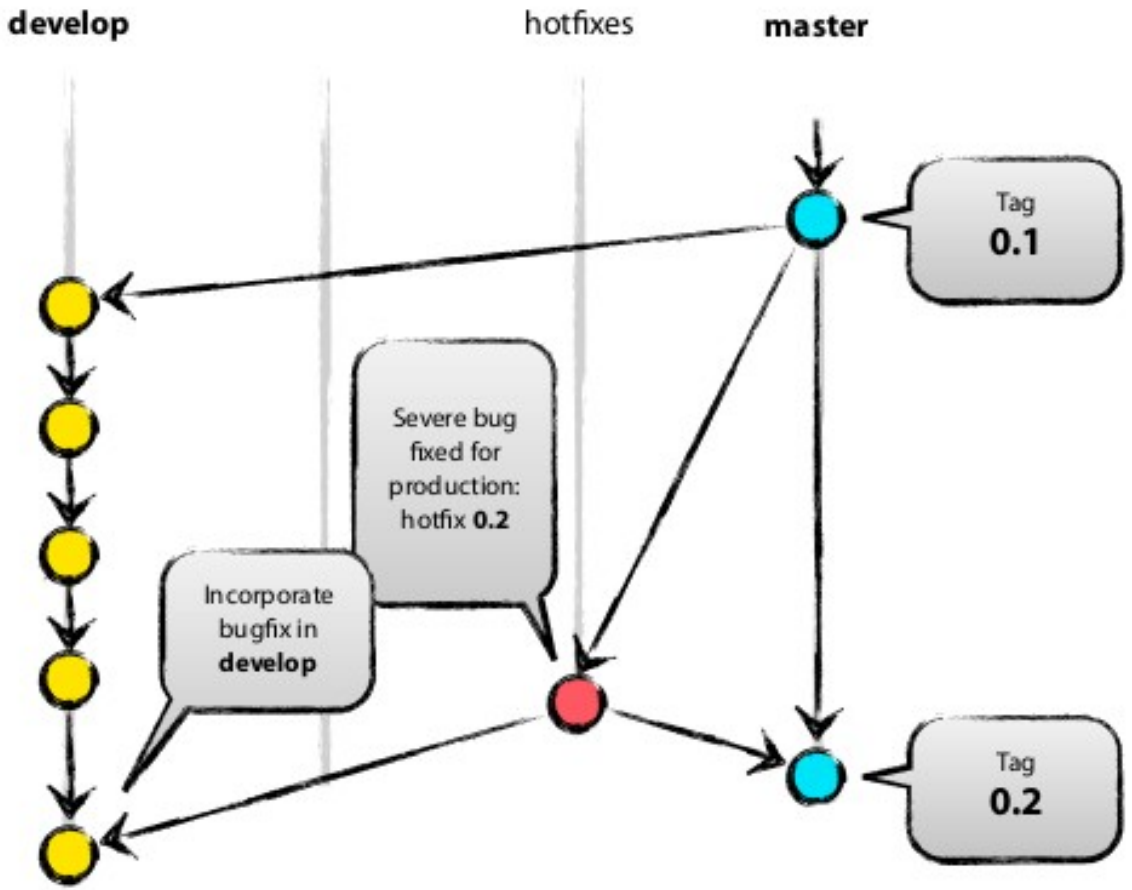
```
git flow feature start feat1
git commit -a
git flow feature start feat2
git commit -a
git commit -a
git flow feature finish feat2
git flow feature feat1
git flow feature rebase
```

3. git-flow: Release vorbereiten




```
git flow release start <release> [<base_on_develop>]
```

3. git-flow: Bugfixes




```
git flow hotfix start <release> [<base>]
git flow hotfix finish <release>
```

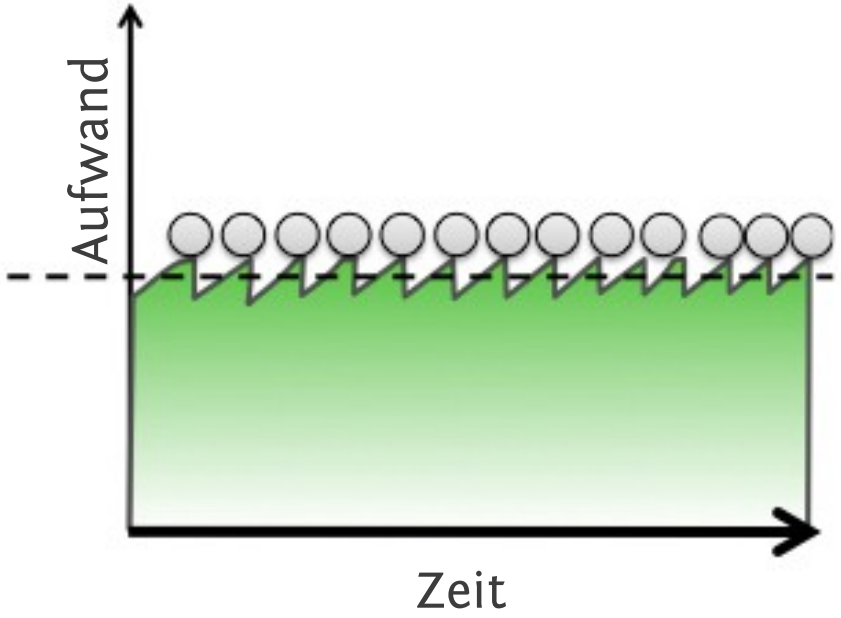
3. Nutzen/Nachteile

+ für größere Teams
+ für lange Release-Zyklen
+ gut dokumentiert, Befehle leicht zu merken

- kein GUI

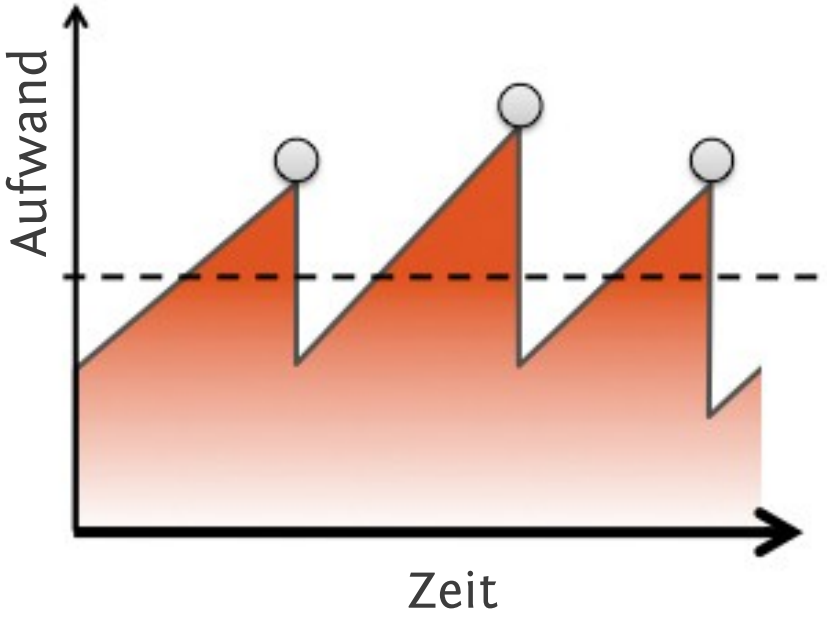
Fall 4. Entwicklung nah an Produktion (DevOps)

Agil:
häufige Releases



Gleichmäßiger Aufwand
Weniger Risiko

Wasserfall:
seltene Releases

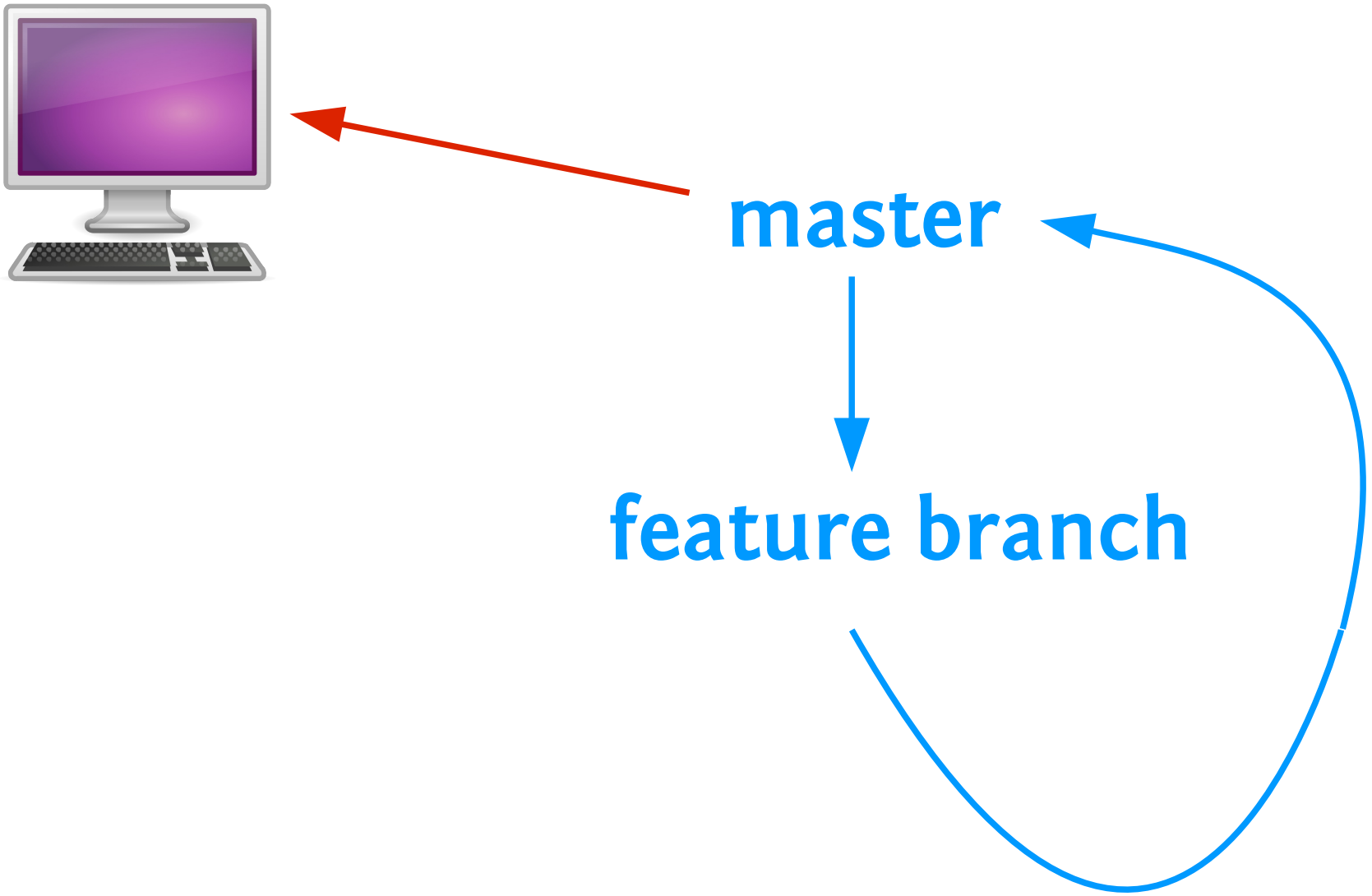


Aufwandspitzen
Hohes Risiko

4. DevOps - der Weg

* schnelle Entwicklung nah an Mainline
* Entwickler sind auch für Produktion verantwortlich
* umfangreiche Test Suites
* hoher Automatisierungsgrad mit CI

4. Release-Zyklus



Angst?

4. Absichern

* nutze Continuous Integration (CI) inkl. Tests auch für Branches!

* nutze Staging Environment (PREPROD)!

* aktiviere nur für Teil der Produktion!

4. Code Review

* pull request (github)

* merge request (email)

4. Vorteile

+ isolierte Feature-Entwicklung
+ kurzes Time-to-market
+ 4-Augen-Prinzip für Go-Live

Tipps und Tricks

gitosis als git Server

gitosis.conf

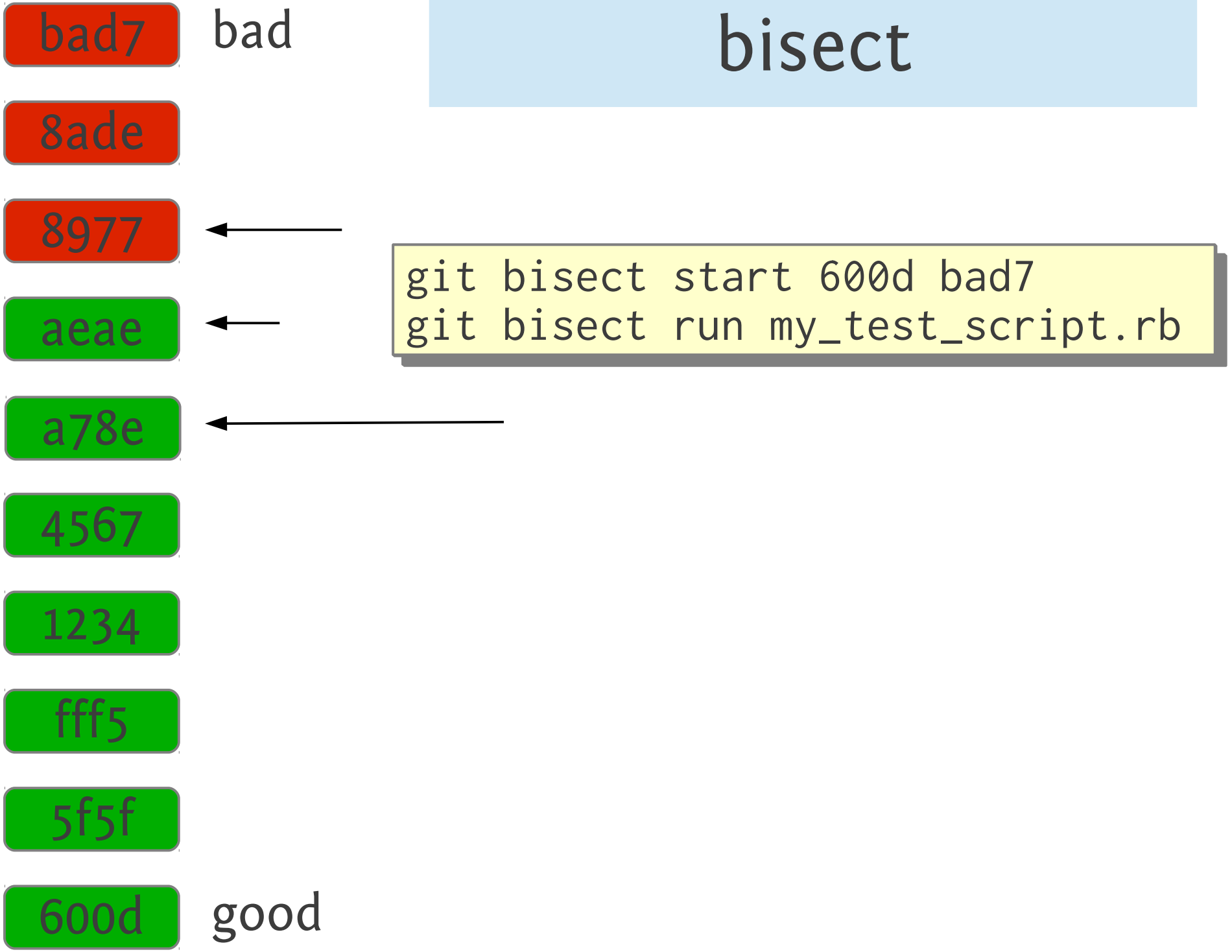
```
[gitosis]

[group vladimir]
writable=*
members=vladimir

[group remote-work]
writable = project1
members = maik vladimir
```

keydir/

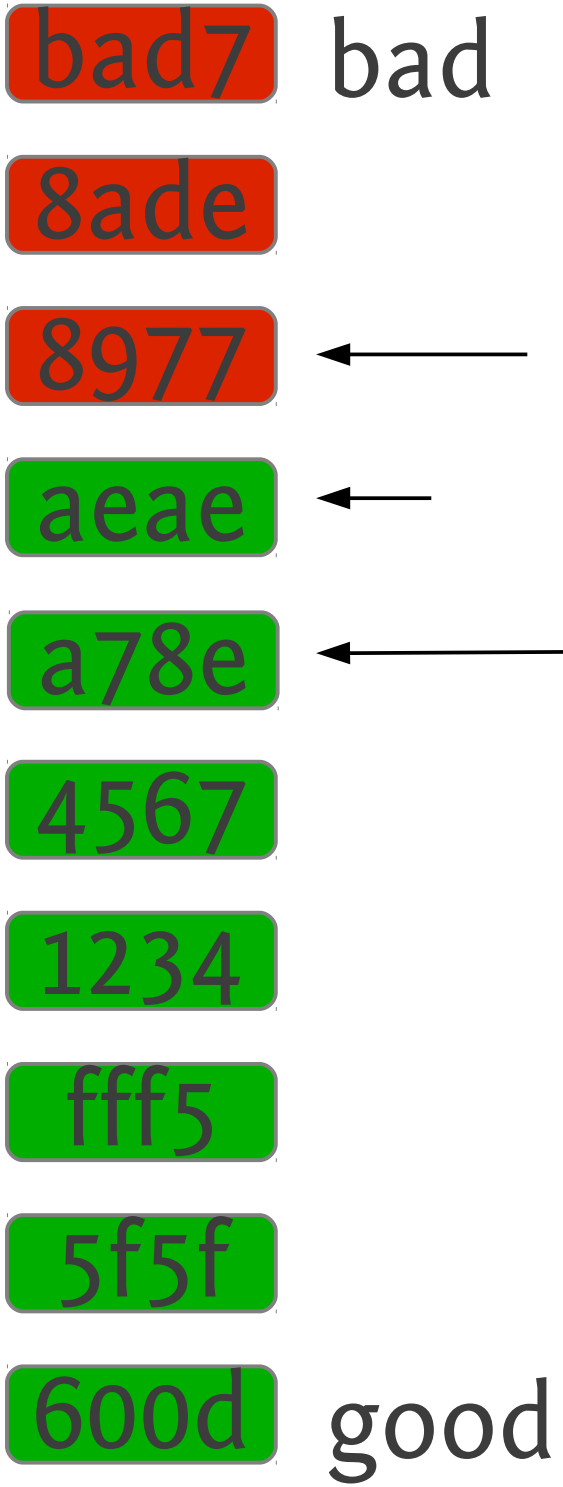
maik.pub

vladimir.pub

bisect


bad7
bad
8ade
8977
aeae
a78e
4567
1234
fff5
5f5f
600d
good
git bisect start 600d bad7
git bisect run my_test_script.rb

bisect




```
git bisect start 600d bad7
git bisect run my_test_script.rb
```

```
bad7 is first bad commit
Author: <vd@example.de>
Date:   Mon Jul 23 20:34:00

Improve performance by
skipping checks

M file1.rb
M file2.java
```


git

Mobile. Web. Consulting. Strategy.

Vladimir Dobriakov

info@mobile-web-consulting.de

Bilder:

- git-flow Diagramm von Vincent Driessen - Creative Commons Lizenz
- Agile-vs-iterative-flow.jpg von Christopher Little via Wikimedia Commons
- Code Review Bild von Damien Hou via flickr.com
- „Faster“ von Alatryste via flickr.com
- die restlichen Bilder lizensiert von iStockphoto.com

Literatur:

- „Pro Git“ von Scott Chacon