3.– 6. September 2012
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Agitation

Warum Sie auf Subversion verzichten sollten

Lars Hupel

Agi·ta·ti·on, die: *politische* Aufklärung, *politische* Werbung

Fahrplan

Kurzeinführung in Git

Werkzeuge für Entwickler

Repository-Management

Workflows

chi wai lau
@tabqwerty

"git gets easier once you get the basic idea that branches are homeomorphic endofunctors mapping submanifolds of a Hilbert space."

http://twitter.com/tabqwerty/statuses/45611899953491968

Zielgruppe

- SVN-Anwender
- Git-Neulinge

Viele Konzepte sind auch auf Mercurial/Bazaar/... übertragbar.
aber: Git hat sehr gute Tool-Unterstützung

Zielgruppe

- SVN-Anwender
- Git-Neulinge

Viele Konzepte sind auch auf Mercurial/Bazaar/... übertragbar.
aber: Git hat sehr gute Tool-Unterstützung

Umfrage

1. Wer hat schon mal ein DVCS genutzt?
2. Wer hat schon mal Git genutzt?
3. Wer nutzt Git im Team?
4. Wer nutzt GitHub?

Umfrage

1. Wer hat schon mal ein DVCS genutzt?
2. Wer hat schon mal Git genutzt?
3. Wer nutzt Git im Team?
4. Wer nutzt GitHub?

Umfrage

1. Wer hat schon mal ein DVCS genutzt?
2. Wer hat schon mal Git genutzt?
3. Wer nutzt Git im Team?
4. Wer nutzt GitHub?

Umfrage

1. Wer hat schon mal ein DVCS genutzt?
2. Wer hat schon mal Git genutzt?
3. Wer nutzt Git im Team?
4. Wer nutzt GitHub?

Git

- dezentralisiert
- schnell
- beliebige Workflows
- erweiterbar

Git

- dezentralisiert
  - jeder hat vollständige Kopie der Geschichte
  - einfaches Branching/Merging
- schnell
- beliebige Workflows
- erweiterbar

Git

- dezentralisiert
  - jeder hat vollständige Kopie der Geschichte
  - einfaches Branching/Merging
- schnell
  - Operationen arbeiten auf lokaler Datenbank
  - `log`, `diff`, `commit` ohne Verzögerung
- beliebige Workflows

- erweiterbar

Git

- dezentralisiert
  - jeder hat vollständige Kopie der Geschichte
  - einfaches Branching/Merging
- schnell
  - Operationen arbeiten auf lokaler Datenbank
  - `log`, `diff`, `commit` ohne Verzögerung
- beliebige Workflows
  - Code-Review, CI
- erweiterbar

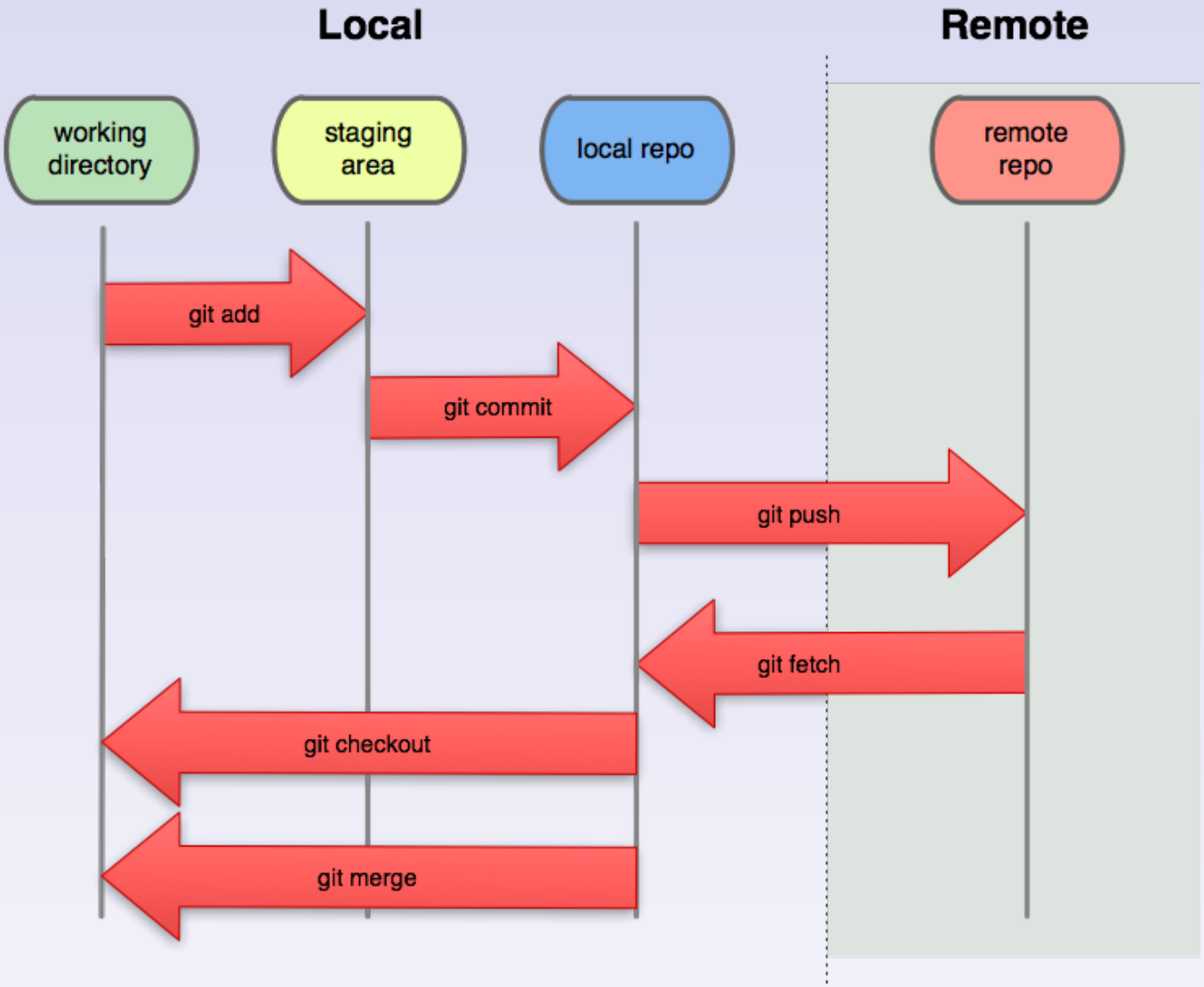
Git

- dezentralisiert
  - jeder hat vollständige Kopie der Geschichte
  - einfaches Branching/Merging
- schnell
  - Operationen arbeiten auf lokaler Datenbank
  - `log`, `diff`, `commit` ohne Verzögerung
- beliebige Workflows
  - Code-Review, CI
- erweiterbar
  - bietet skriptbare Low-Level-Operationen
  - einige Kommandos arbeiten (semi-)automatisch

Lokaler Workflow

1. Änderungen von einem Repository ins eigene übernehmen
2. Arbeitskopie verändern
3. Änderungen in den *Index* übertragen
4. Index committen
5. Änderungen vom eigenen zu einem Repository publizieren

Lokaler Workflow



Git Book by Scott Chacon, `http://book.git-scm.com/`

Gängige Vorbehalte

- sich abspaltende Codebasis führt zu Kontrollverlust
- kompliziertes Erlernen
- divergierende Revisionsnummern
- Berechtigungen?

Gängige Vorbehalte

- sich abspaltende Codebasis führt zu Kontrollverlust
  exzellente Tool-Unterstützung
- kompliziertes Erlernen
- divergierende Revisionsnummern
- Berechtigungen?

Gängige Vorbehalte

- sich abspaltende Codebasis führt zu Kontrollverlust
  exzellente Tool-Unterstützung
- kompliziertes Erlernen
  mittlerweile einfacher zu lernen als Subversion
- divergierende Revisionsnummern

- Berechtigungen?

Gängige Vorbehalte

- sich abspaltende Codebasis führt zu Kontrollverlust
  exzellente Tool-Unterstützung
- kompliziertes Erlernen
  mittlerweile einfacher zu lernen als Subversion
- divergierende Revisionsnummern
  Namen statt Nummern
- Berechtigungen?

Gängige Vorbehalte

- sich abspaltende Codebasis führt zu Kontrollverlust
  exzellente Tool-Unterstützung
- kompliziertes Erlernen
  mittlerweile einfacher zu lernen als Subversion
- divergierende Revisionsnummern
  Namen statt Nummern
- Berechtigungen?
  Einsatz externer Tools

Objektmodell

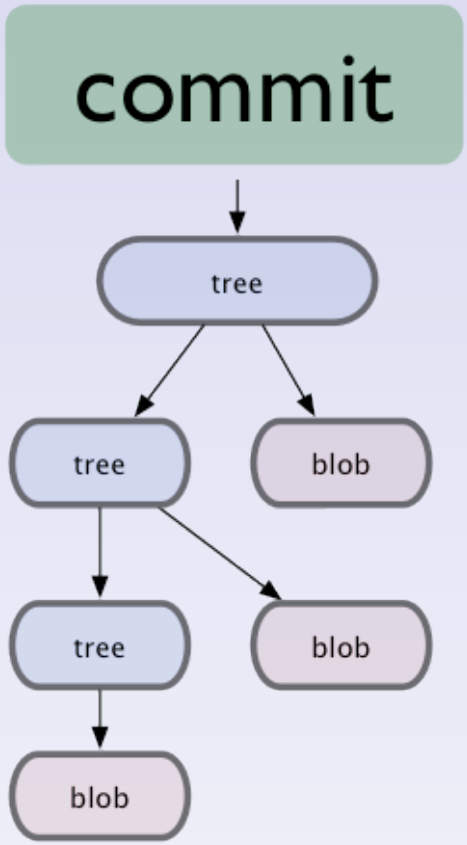
Git verwaltet

- *Objekte*

  Blobs, Trees, Commits, Tags

- und *Referenzen*

  Branches, Tags



Git Book

Branches

oder: Warum sind Branches so günstig?

```
$ git checkout -b release-2.0
```

- legt neue Referenz an
- setzt HEAD auf neue Branch

```
$ git commit
```

- legt neuen Commit an
- setzt Branch auf neuen Commit

Branches

Branch early, branch often

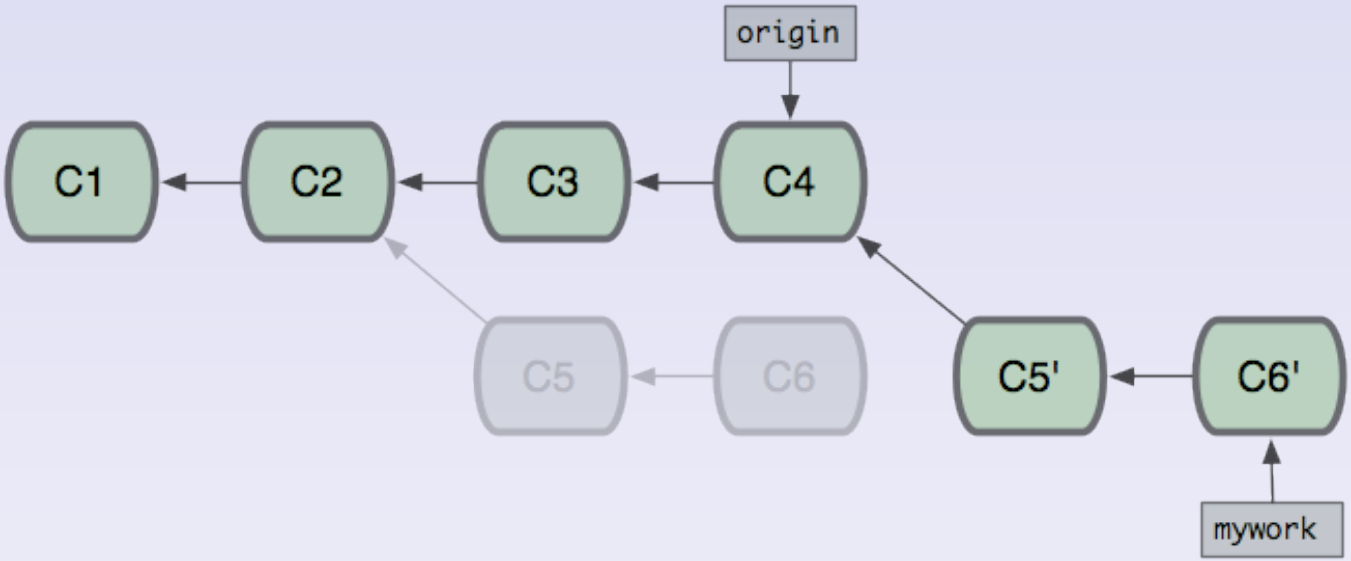
- `3.14.x`
- `issue/42`
- `feature/pacman`
- `custom/zcorp`

Porcelain (high level)

Git bietet eine Fülle an Werkzeugen:

- `rebase [--interactive]`
- `add --patch`
- `stash`
- `bisect`
- `cherry-pick`, `cherry`

Rebase



Git Book

Merge vs. Rebase

- Merge: vereinigt zwei divergierende Stränge
- Rebase: aktualisiert einen Strang auf eine neue Basis

Merge vs. Rebase

- Merge: vereinigt zwei divergierende Stränge
  - Fertigstellung von zusammenhängenden Änderungen
  - ermöglicht „Octopus“
- Rebase: aktualisiert einen Strang auf eine neue Basis

Merge vs. Rebase

- Merge: vereinigt zwei divergierende Stränge
  - Fertigstellung von zusammenhängenden Änderungen
  - ermöglicht „Octopus“
- Rebase: aktualisiert einen Strang auf eine neue Basis
  - bei langer Arbeit an einem Feature
  - verringert das Konfliktrisiko

Merge vs. Rebase

Anti-Beispiel

```
| * | | | | eb57398 adding fuse stuff to .gitignore
| * | | | | 5369fa6 Initial version of Scanner + minor changes at CamlLightSp
| * | | | | f866afd parts of CamlLightSpec
* | | | | |   27d7c78 Merge branch 'master' of vmschlichter9.informatik.tu-mu
|\ \ \ \ \ \
| * | | | | | 53f8f72 Fixed error in Match expression. Added more tests.
| * | | | | | dfd50a8 removed head entry
| * | | | | |   4c037fd Merge branch 'master' of vmschlichter9.informatik.tu-
| |\ \ \ \ \ \
| * | | | | | | 52725fa added junit tests
| | |_|_|_|/ /
| |/| | | | |
* | | | | | | 630177b Tuple code generations works (with a more sophisticatec
| |/ / / / /
|/| | | | |
* | | | | |   1d3761d Merge branch 'master' of vmschlichter9.informatik.tu-mu
|\ \ \ \ \ \
| | |_|_|_|/
| |/| | | |
| * | | | | c289fdb utility functions
| * | | | | 8e055b2 notabbing
```

Interaktives Rebase

oder: Let’s fix the history

1. Team arbeitet an Feature
2. Feature fertiggestellt
3. soll gemerged werden, aber: History unsauber
   - versehentlich zu wenig comitted
   - falsche Dateiberechtigungen
   - ...

Interaktives Rebase

```
$ git rebase -i <commit>^
```

Möglichkeiten:

`reword`	Commit-Message bearbeiten
`edit`	Commit nachbearbeiten
`squash`	zwei oder mehr Commits verschmelzen
`exec`	Shell-Kommando ausführen
`-`	Commit „löschen“

Interaktives Rebase

Demo

Exkurs: Datensicherheit

With great power comes great responsibility

- Git löscht Objekte aus der Datenbank nicht (sofort)
- aber: “dangling objects” können entstehen
- Abhilfe: `git fsck --lost-found` und `git show`
- besser: vor Rebase branchen
- außerdem: Git-Meldungen genau lesen

Exkurs: Datensicherheit

With great power comes great responsibility

- Git löscht Objekte aus der Datenbank nicht (sofort)
- aber: “dangling objects” können entstehen
- Abhilfe: `git fsck --lost-found` und `git show`
- besser: vor Rebase branchen
- außerdem: Git-Meldungen genau lesen

Exkurs: Datensicherheit

With great power comes great responsibility

- Git löscht Objekte aus der Datenbank nicht (sofort)
- aber: “dangling objects” können entstehen
- Abhilfe: `git fsck --lost-found` und `git show`
- besser: vor Rebase branchen
- außerdem: Git-Meldungen genau lesen

Exkurs: Datensicherheit

Demo

Patches hinzufügen

oder: Die guten ins Töpfchen

```
$ git add --patch .
```

- Problem: innerhalb einer Datei wurden mehrere unabhängige Änderungen gemacht
- interaktiver Modus ermöglicht feingranulare Selektion
- bei Bedarf kann manuell editiert werden
- Vorsicht bei zu kleinen Änderungen
  ⇝ potenzielle Build-Probleme

Patches hinzufügen

oder: Die guten ins Töpfchen

```
$ git add --patch .
```

- Problem: innerhalb einer Datei wurden mehrere unabhängige Änderungen gemacht
- interaktiver Modus ermöglicht feingranulare Selektion
- bei Bedarf kann manuell editiert werden
- Vorsicht bei zu kleinen Änderungen
  $\rightsquigarrow$ potenzielle Build-Probleme

Patches hinzufügen

Demo

Stash

oder: Ständig wird man unterbrochen...

- Problem: während der Arbeit an X muss dringend Y gemacht werden
- eigentliches Problem: Verwaltung verschiedener unabhängiger Arbeitskopien
- im Stash können Änderungen temporär abgelegt werden

Stash

Demo

Bisect

oder: Wer ist es gewesen?

- Regression tritt auf – aber seit wann?
- Markieren eines beliebigen „guten“ Zustands
- Markieren eines beliebigen „schlechten“ Zustands
- `git bisect` ⇝ Binärsuche

Bisect

oder: Wer ist es gewesen?

- Regression tritt auf – aber seit wann?
- Markieren eines beliebigen „guten“ Zustands
- Markieren eines beliebigen „schlechten“ Zustands
- `git bisect` $\rightsquigarrow$ Binärsuche

Bisect

Demo

Cherry(-pick)

- Problem: Portieren von Patches auf einen älteren Zweig
- außerdem: Welche Patches wurden schon portiert?

Tomasz Sienicki,
http://commons.wikimedia.org/wiki/File:Kt_xx_0011_ubt.jpeg

Cherry(-pick)

Demo

Grundregeln

Subversion

- zentraler Server
- ein Repo für alle Projekte
- `svn:externals`
- Commit Hooks
- Path permissions

Git

- ein Repo pro Aufgabe
- ein Repo pro Projekt
- `git submodule`
- Update, Commit, ... Hooks
- Gitolite/Gitosis

Gitolite

- SSH + Git + Rechteverwaltung
- Authentifizierung: Public key
- Authorisierung: konfigurierbar
- aktiv entwickelt:
  `https://github.com/sitaramc/gitolite`
- ausführliche Dokumentation

Gitolite

Konfigurationsbeispiel

```
repo gitolite-admin
    RW+ = admin
repo project
    RW+ = admin
    RW master$ = @developers
    R integration$ refs/tags/v.* = @qa
```

Gitolite

Berechtigungen

... für

Read, Write, Rewind, Create, Delete, Merge

... auf

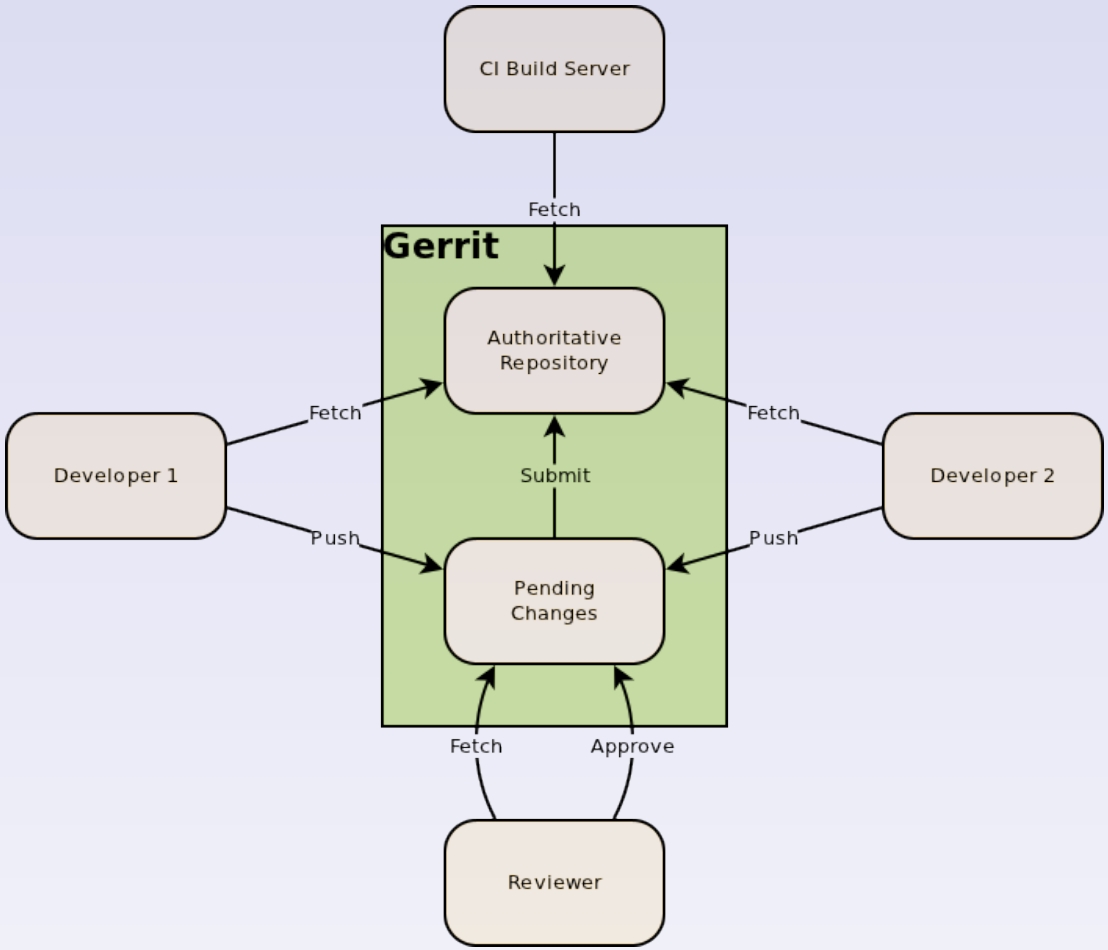
Branches, Tags, Trees

Gerrit

- webbasierter Code Review für Git
- außerdem Repo-Verwaltung
- Integration mit Jenkins
- Submit via Web
- aktiv entwickelt: `http://code.google.com/p/gerrit/`

Gerrit

Übersicht



http://gerrit.googlecode.com/git/Documentation/images/intro-quick-central-gerrit.png

Gerrit

Workflow

1. Commit zu Gerrit pushen
2. Gerrit triggert Jenkins-Build
3. Code Review, evtl. Nachbesserung
4. Gerrit merged und übernimmt den Commit

Gerrit

Demo

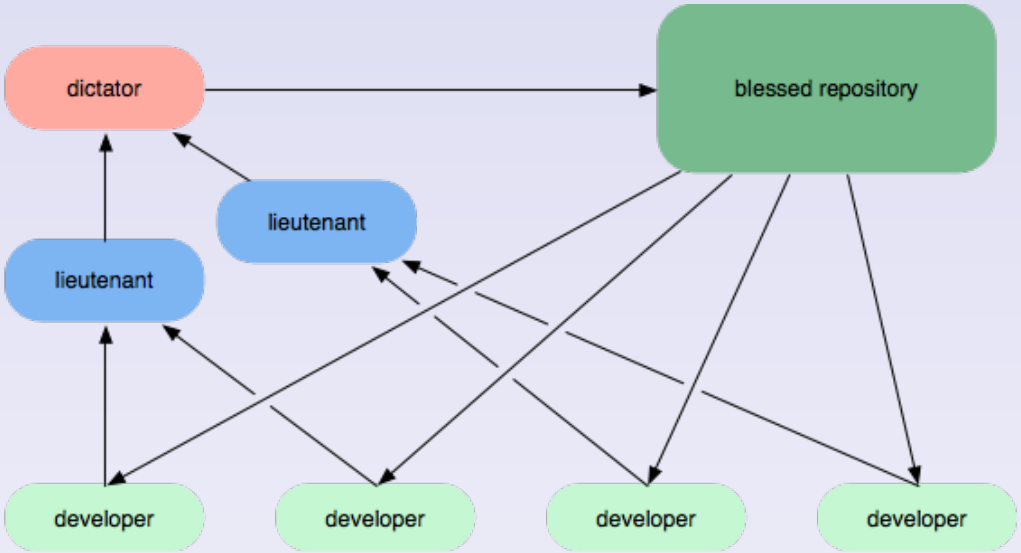
Bugs Everywhere

- verteiltes Bug-Tracking-System
- verwaltet Bugs als Textdateien im Repo
- interessante Idee, aber nicht praxisfähig

Beispiel: Linux-Kernel

- aufgeteilt in mehrere Subsysteme
- jedes Subsystem hat eigenen Maintainer
- Änderungen werden vom Maintainer geprüft
- Linus übernimmt Änderungen gesammelt

Beispiel: Linux-Kernel

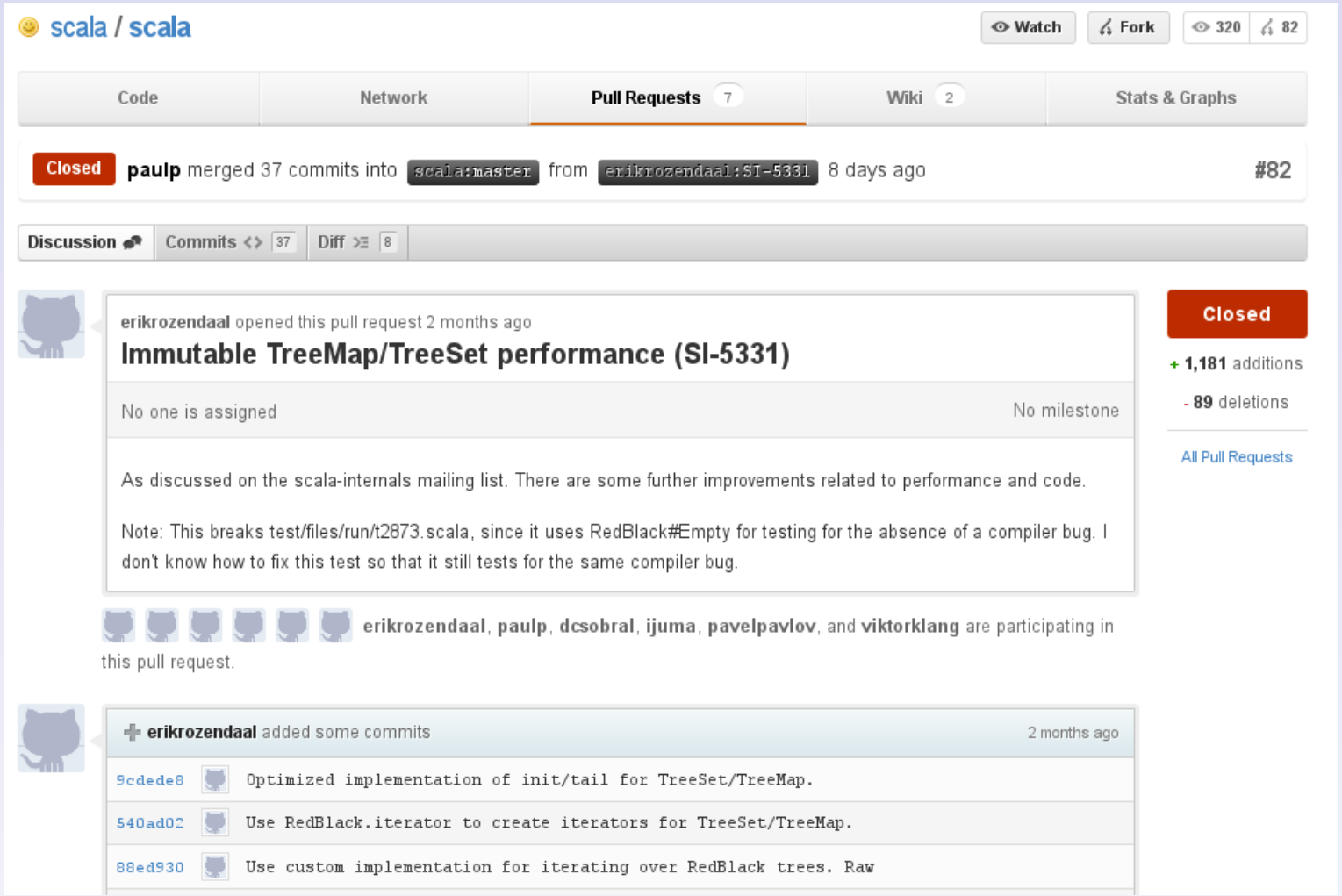


Git Book

Beispiel: GitHub

- Repos werden „geforkt“
- Änderungen in separate Branch pushen
- Pull Request öffnen
- Owner kann Pull Request automatisiert übernehmen

Beispiel: GitHub



https://github.com/scala/scala/pull/82

Beispiel: Gitflow

- Definition von Branches mit festgelegten Rollen

`master`	stabiler Code (“production ready”)
`develop`	Code für das nächste Release (“integration”)
`feature/*`	unabhängiges Feature
`release/*`	unmittelbare Release-Vorbereitungen

- Vorteil: Tool-Unterstützung

Nachteile von Git

```
$ man git<TAB><TAB>
Display all 166 possibilities? (y or n)
```

Fragen?

larsrh.github.com
lars.hupel@tum.de