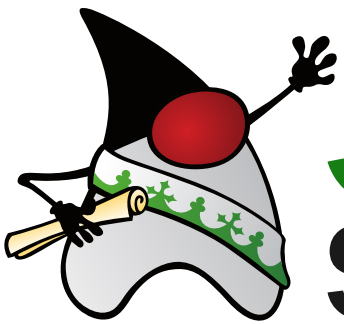
Verteilte Versionsverwaltung mit Git

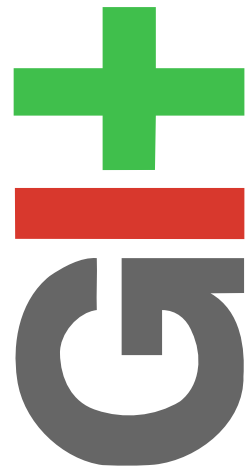
Jan Dittberner

Communardo Software GmbH, Dresden

15.09.2011



JUG saxony



Einleitung

Über den Vortragenden

Auswertung Vorbefragung

Communardo Software GmbH

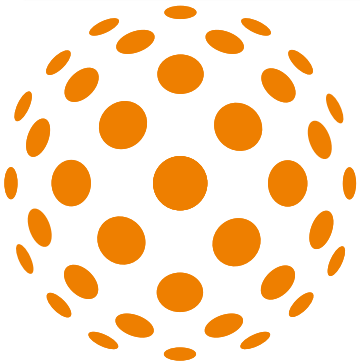
- Anbieter von Softwarelösungen zur Optimierung der Kommunikation und Zusammenarbeit in Teams, Projekten und Unternehmen
- Software- und Beratungshaus für gesamten Realisierungszyklus

Communardo Software GmbH

Consulting (Strategie-, Technologie-, Einführungsberatung)		Team Collaboration					
		Knowledge Management					
		Enterprise 2.0 Solutions					
Software Solutions		Portal-lösungen	Enterprise Mashups	Social Intranet / Intranet 2.0	Enterprise Search	Issue Management	Identity Management
Software Engineering	IBM Business Partner	Websphere	Mashup Center	Lotus Quickr & Connections	Ontoprise / Omnifind		Tivoli
	Microsoft GOLD CERTIFIED Partner	SharePoint		SharePoint	FAST Search Server 2010	SharePoint	
	ATLASSIAN PARTNER			Confluence		J IRA	Crowd
	Open Source	Liferay Portal		Wordpress			CAS
Infrastructure Services		Lizenzierung, Installation, Application Management, 2nd-Level Support					

Jan Dittberner

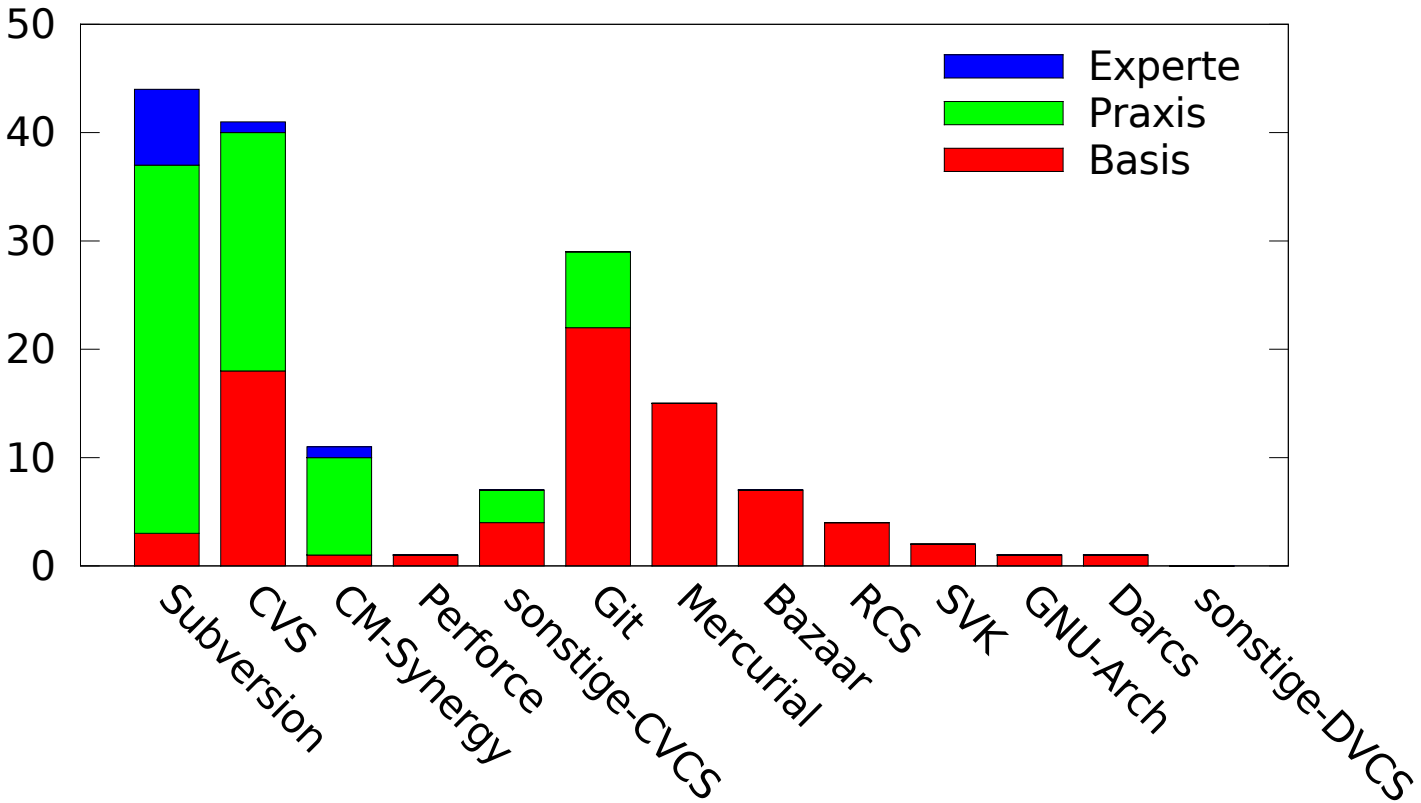
- Softwarearchitekt
- Debian Developer
- CAcert.org Infrastrukturadministrator
- Spezialgebiete: freie Software, Kryptografie, ...



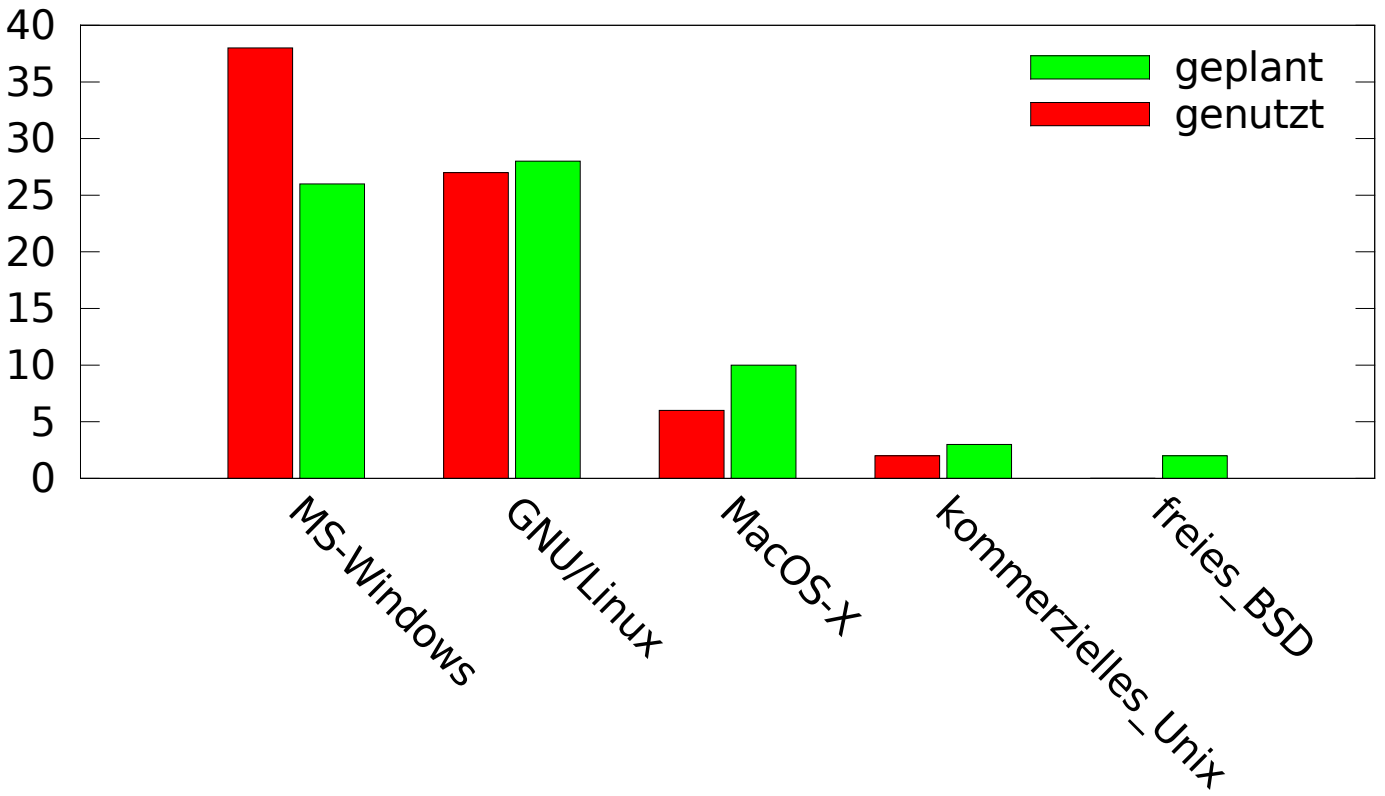
Jan Dittberner

- Softwarearchitekt
- Debian Developer
- CAcert.org Infrastrukturadministrator
- Spezialgebiete: freie Software, Kryptografie, ...
- verheiratet, 3 Kinder, eine Katze ☺

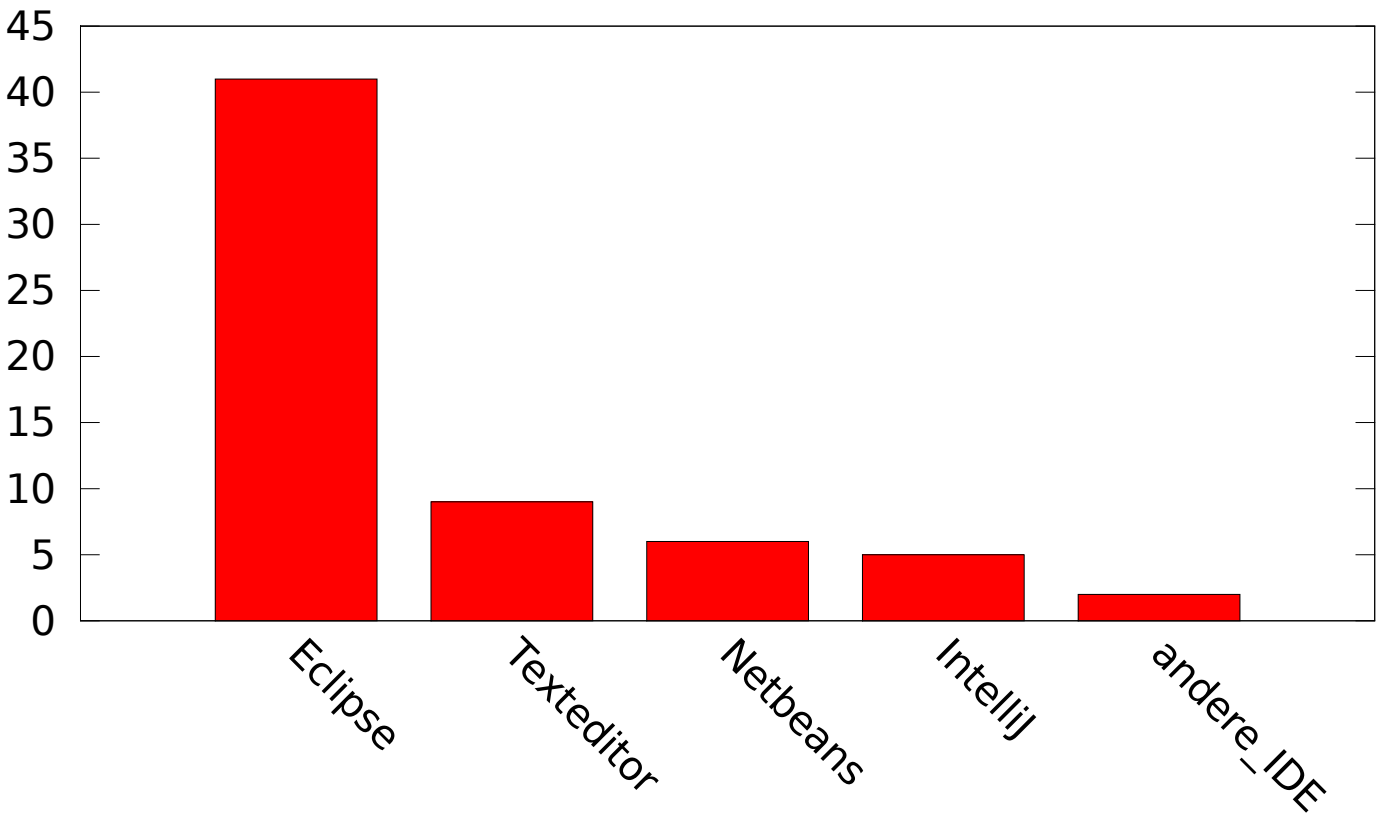
Bekanntheitsgrad von Versionskontrollsystemen



verwendete/geplante Betriebssysteme



verwendete IDEs



gewünschte Themen

95%	Umgang mit Branches, Beispielworkflow
85%	Einführung in Git
79%	Nutzung von Git im Zusammenspiel mit Subversion
76%	IDE-Unterstützung für Git
73%	Integration von Git mit CI-Tools wie Jenkins
71%	Vergleich verteilte vs. zentrale Versionskontrolle
71%	Integration von Git mit Issue-Tracking-Systemen
68%	Integration von Git mit Maven
53%	Gegenüberstellung von SVN- mit Git-Kommandos

Verteilte VCS/Git

- Zentrale vs. verteilte VCS
- Git speziell
- Erste Schritte mit Git

Zentral	Verteilt
ein zentrales Repository (evtl. repliziert)	

Zentral	Verteilt
ein zentrales Repository (evtl. repliziert)	beliebig viele (gleichberechtigte) Klone

Zentral	Verteilt
ein zentrales Repository (evtl. repliziert)	beliebig viele (gleichberechtigte) Klone
lokale Working Copy (meist) ohne Historie	

Zentral	Verteilt
ein zentrales Repository (evtl. repliziert)	beliebig viele (gleichberechtigte) Klone
lokale Working Copy (meist) ohne Historie	(komplette) lokale Historie

Zentral	Verteilt
ein zentrales Repository (evtl. repliziert)	beliebig viele (gleichberechtigte) Klone
lokale Working Copy (meist) ohne Historie	(komplette) lokale Historie
viele Remote-Operationen	

Zentral	Verteilt
ein zentrales Repository (evtl. repliziert)	beliebig viele (gleichberechtigte) Klone
lokale Working Copy (meist) ohne Historie	(komplette) lokale Historie
viele Remote-Operationen	alles außer Repositoryabgleich ist lokal

Zentral	Verteilt
Austausch nur über zentrales Repository	

Zentral	Verteilt
Austausch nur über zentrales Repository	Austausch zwischen beliebigen Klonen möglich

Zentral	Verteilt
Austausch nur über zentrales Repository	Austausch zwischen beliebigen Klonen möglich
Netzwerk erforderlich	

Zentral	Verteilt
Austausch nur über zentrales Repository	Austausch zwischen beliebigen Klonen möglich
Netzwerk erforderlich	Netzwerk nur für Repository-abgleich nötig

Zentral	Verteilt
Austausch nur über zentrales Repository	Austausch zwischen beliebigen Klonen möglich
Netzwerk erforderlich	Netzwerk nur für Repository-abgleich nötig
(meist) starre vom Tool vorgegebene Workflows	

Zentral	Verteilt
Austausch nur über zentrales Repository	Austausch zwischen beliebigen Klonen möglich
Netzwerk erforderlich	Netzwerk nur für Repository-abgleich nötig
(meist) starre vom Tool vorgegebene Workflows	beliebige Workflows (später mehr dazu)

Warum Git und nicht ...?

- leichtgewichtige lokale Branches

Warum Git und nicht ...?

- leichtgewichtige lokale Branches
- sehr schnell

Warum Git und nicht ...?

- leichtgewichtige lokale Branches
- sehr schnell
- die Staging-Area (Index)

Warum Git und nicht ...?

- leichtgewichtige lokale Branches
- sehr schnell
- die Staging-Area (Index)
- Speicherplatz- und Bandbreitensparsamkeit

Warum Git und nicht ...?

- leichtgewichtige lokale Branches
- sehr schnell
- die Staging-Area (Index)
- Speicherplatz- und Bandbreitensparsamkeit
- Rebasing um lokale Änderungen zusammenzufassen

Warum Git und nicht ...?

- leichtgewichtige lokale Branches
- sehr schnell
- die Staging-Area (Index)
- Speicherplatz- und Bandbreitensparsamkeit
- Rebasing um lokale Änderungen zusammenzufassen
- sehr umfangreiche Dokumentation, Bücher und Community [1]

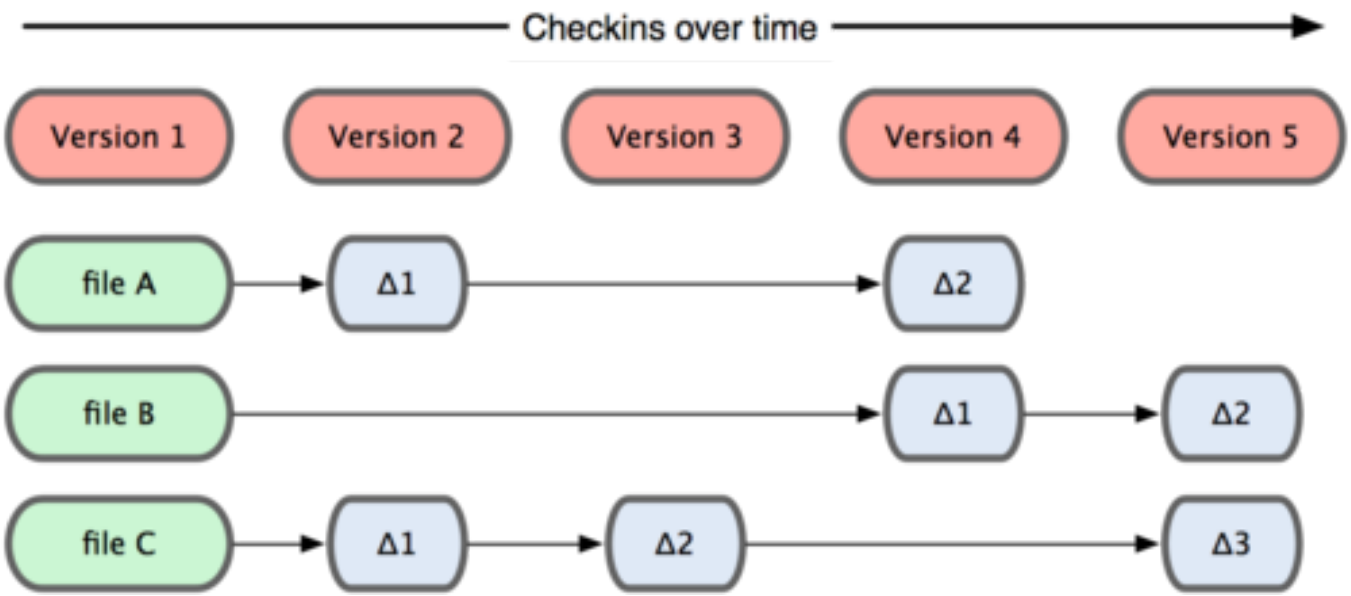
Warum Git und nicht ...?

- leichtgewichtige lokale Branches
- sehr schnell
- die Staging-Area (Index)
- Speicherplatz- und Bandbreitensparsamkeit
- Rebasing um lokale Änderungen zusammenzufassen
- sehr umfangreiche Dokumentation, Bücher und Community [1]
- `http://de.whygitisbetterthanx.com/`

Unterschied zu den meisten anderen VCS

- Die meisten VCS speichern Änderungen zu einer Basisversion

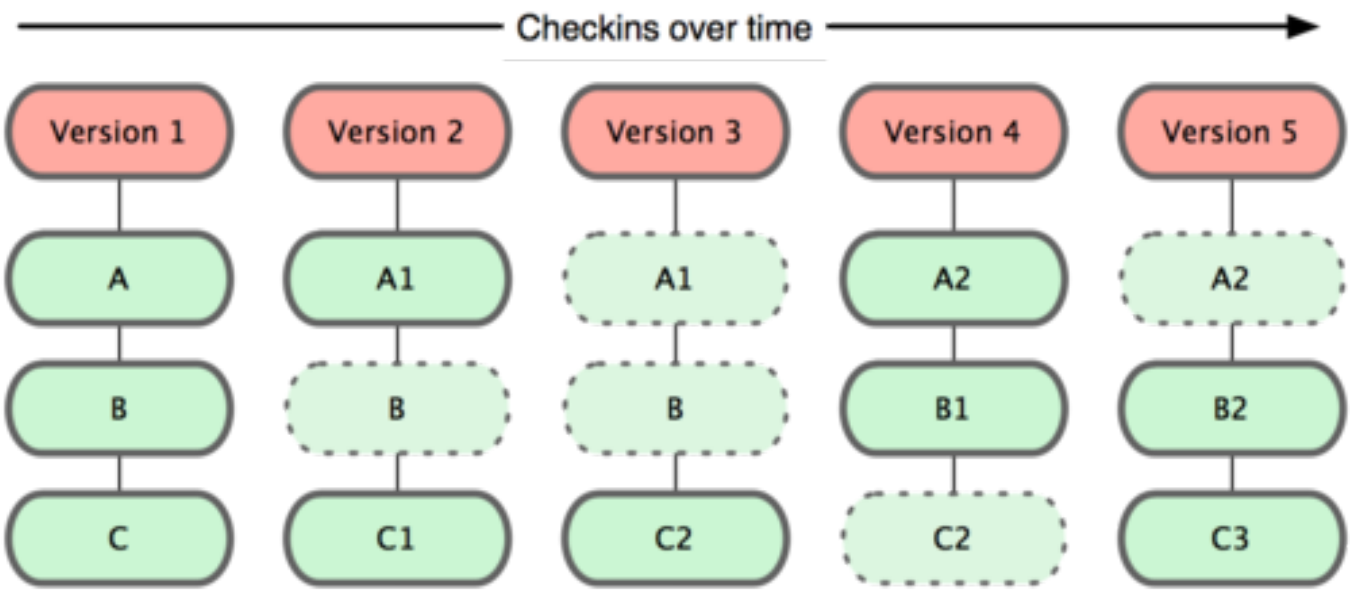
Bildquelle: `http://progit.org` [2]



Unterschied zu den meisten anderen VCS

- Die meisten VCS speichern Änderungen zu einer Basisversion
- Git speichert Datenschnappschüsse und keine Deltas

Bildquelle: `http://progit.org` [2]



Objekte in Git

- Blob – Dateiinhalte

Objekte in Git

- Blob – Dateiinhalte
- Tree – Zeiger auf Dateiinhalte und Trees

Objekte in Git

- Blob – Dateiinhalte
- Tree – Zeiger auf Dateiinhalte und Trees
- Commit – Zeiger auf Version eines Trees

Objekte in Git

- Blob – Dateiinhalte
- Tree – Zeiger auf Dateiinhalte und Trees
- Commit – Zeiger auf Version eines Trees
- Tag – Name für einen bestimmten Commit

Objekte in Git

- Blob – Dateiinhalte
- Tree – Zeiger auf Dateiinhalte und Trees
- Commit – Zeiger auf Version eines Trees
- Tag – Name für einen bestimmten Commit
- Branch – Entwicklungszweig mit eigenem HEAD

Objekte in Git

- Blob – Dateiinhalte
- Tree – Zeiger auf Dateiinhalte und Trees
- Commit – Zeiger auf Version eines Trees
- Tag – Name für einen bestimmten Commit
- Branch – Entwicklungszweig mit eigenem HEAD

- Blobs, Trees und Commits können eindeutig und fälschungssicher über SHA-1-Hash identifiziert werden

Objekte in Git

- Blob – Dateiinhalte
- Tree – Zeiger auf Dateiinhalte und Trees
- Commit – Zeiger auf Version eines Trees
- Tag – Name für einen bestimmten Commit
- Branch – Entwicklungszweig mit eigenem HEAD

- Blobs, Trees und Commits können eindeutig und fälschungssicher über SHA-1-Hash identifiziert werden
- Tags und Branches sind Namen für Commit-Hashes

Drei Bäume in Git

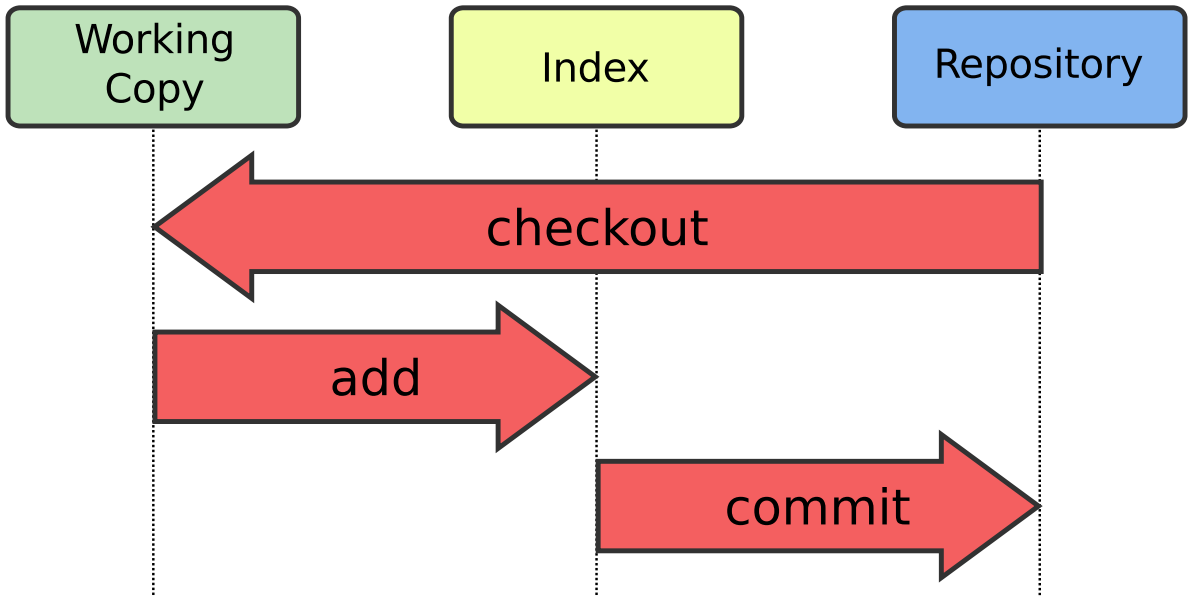
- HEAD – letzter Commit, Parent des nächsten Commits

Drei Bäume in Git

- HEAD – letzter Commit, Parent des nächsten Commits
- Index – Vorbereitung für nächsten Commit

Drei Bäume in Git

- HEAD – letzter Commit, Parent des nächsten Commits
- Index – Vorbereitung für nächsten Commit
- Working Copy – Arbeitsverzeichnis



Git installieren

Linux

- Debian/Ubuntu

```
sudo aptitude install git-all
```

Git installieren

Linux

- Debian/Ubuntu

```
sudo aptitude install git-all
```

- CentOS/RHEL (mit EPEL [3]) oder Fedora

```
yum install git gitk
```

Git installieren

Linux

- Debian/Ubuntu

```
sudo aptitude install git-all
```

- CentOS/RHEL (mit EPEL [3]) oder Fedora

```
yum install git gitk
```

Windows

- msysgit [4] oder git-Paket für Cygwin [5]
- TortoiseGit [6] (benötigt msysGit)

Bei Git bekanntmachen

```
git config --global user.name "Jan Dittberner"
git config --global user.email ↵
    "jan.dittberner@communardo.de"
```

Bei Git bekanntmachen

```
git config --global user.name "Jan Dittberner"
git config --global user.email ↵
    "jan.dittberner@communardo.de"
```

Aktuelles Verzeichnis zu Git-Repository machen

```
git init
```

Datei in Index aufnehmen

```
git add newfile.txt
```

Datei in Index aufnehmen

```
git add newfile.txt
```

Änderungen (lokal) einchecken

```
git commit
```

Branch `featureX` erzeugen und HEAD auf Branch setzen

```
git checkout -b featureX
```

Branch `featureX` erzeugen und HEAD auf Branch setzen

```
git checkout -b featureX
```

Branch `featureY` in Working Copy auschecken

```
git checkout featureY
```

Branch `featureX` erzeugen und HEAD auf Branch setzen

```
git checkout -b featureX
```

Branch `featureY` in Working Copy auschecken

```
git checkout featureY
```

Branch erzeugen ohne Auschecken

```
git branch featureZ
```

(Remote-)Repository klonen

```
git clone gitolite@gitserver:reponame
```

(Remote-)Repository klonen

```
git clone gitolite@gitserver:reponame

git clone $HOME/gitrepos/project.git
```

(Remote-)Repository klonen

```
git clone gitolite@gitserver:reponame

git clone $HOME/gitrepos/project.git

git clone git://github.com/jandd/deform.git
```

Unterstützte Protokolle: ssh, git, http(s), ftp(s), rsync, file

Änderungen aus Remote-Repository holen

```
git fetch
```

Änderungen aus Remote-Repository holen

```
git fetch
```

inklusive Merge in Working Copy ...

```
git pull
```

Änderungen aus Remote-Repository holen

```
git fetch
```

inklusive Merge in Working Copy ...

```
git pull
```

Änderungen in Remote-Repository übertragen

```
git push
```

Weiteres Remote-Repository hinzufügen

```
git remote add kollege ↵
    http://colleaguesmachine/repo.git
```

Weiteres Remote-Repository hinzufügen

```
git remote add kollege ↵
    http://colleaguesmachine/repo.git
```

das Standardrepository heißt `origin`

Weiteres Remote-Repository hinzufügen

```
git remote add kollege ↩
    http://colleaguesmachine/repo.git
```

das Standardrepository heißt `origin`

von Remote-Repository aktualisieren

git fetch kollege

History ansehen

```
git log
```

History ansehen

```
git log
```

Status der Working Copy und des Index

```
git status
```

History ansehen

```
git log
```

Status der Working Copy und des Index

```
git status
```

Unterschied zu Index

```
git diff
```

Änderungen aus Branch in aktuellen Branch mergen

```
git merge featureX
```

Änderungen aus Branch in aktuellen Branch mergen

```
git merge featureX
```

Hilfe zu <command>

```
git help <command>
```

Von Subversion zu Git

- Git als Subversion-Frontend
- Umstieg auf Git

Warum Git als Frontend?

- lokal alle Möglichkeiten von Git nutzbar

Warum Git als Frontend?

- lokal alle Möglichkeiten von Git nutzbar
- SVN-Infrastruktur kann weitergenutz werden

Warum Git als Frontend?

- lokal alle Möglichkeiten von Git nutzbar
- SVN-Infrastruktur kann weitergenutz werden
- „sanfter“ Einstieg in DVCS

Warum Git als Frontend?

- lokal alle Möglichkeiten von Git nutzbar
- SVN-Infrastruktur kann weitergenutz werden
- „sanfter“ Einstieg in DVCS

Nachteile gegenüber „nativem“ Git

- eingeschränkte (lokale) Toolunterstützung

Warum Git als Frontend?

- lokal alle Möglichkeiten von Git nutzbar
- SVN-Infrastruktur kann weitergenutz werden
- „sanfter“ Einstieg in DVCS

Nachteile gegenüber „nativem“ Git

- eingeschränkte (lokale) Toolunterstützung
- langsame Remote-Repository-Aktionen

Warum Git als Frontend?

- lokal alle Möglichkeiten von Git nutzbar
- SVN-Infrastruktur kann weitergenutz werden
- „sanfter“ Einstieg in DVCS

Nachteile gegenüber „nativem“ Git

- eingeschränkte (lokale) Toolunterstützung
- langsame Remote-Repository-Aktionen
- lokale Branches werden im SVN nicht als solche getrackt

SVN-Repository als lokales Git-Repository

```
git svn clone -s -A ~/svn-authors.txt ↵
    https://svn.company.org/projectrepo
```

kann bei großen Repositories sehr lange dauern

SVN-Repository als lokales Git-Repository

```
git svn clone -s -A ~/svn-authors.txt ⤶
    https://svn.company.org/projectrepo
```

kann bei großen Repositories sehr lange dauern

lokale Git-Commits in SVN-Repository

```
git svn dcommit
```

SVN-Repository als lokales Git-Repository

```
git svn clone -s -A ~/svn-authors.txt ↵
    https://svn.company.org/projectrepo
```

kann bei großen Repositories sehr lange dauern

lokale Git-Commits in SVN-Repository

```
git svn dcommit
```

neue SVN-Revisions abholen

```
git svn fetch
```

- vorher prüfen ob alle verwendeten Tools Git unterstützen

- vorher prüfen ob alle verwendeten Tools Git unterstützen
- SVN-Git-Clone mit vollständiger Autoren-Datei

- vorher prüfen ob alle verwendeten Tools Git unterstützen
- SVN-Git-Clone mit vollständiger Autoren-Datei
- Aufbereitung für Git-Spezialitäten (Tags sind keine Kopien, svn:ignore zu .gitignore, ...)

- vorher prüfen ob alle verwendeten Tools Git unterstützen
- SVN-Git-Clone mit vollständiger Autoren-Datei
- Aufbereitung für Git-Spezialitäten (Tags sind keine Kopien, svn:ignore zu .gitignore, ...)
- gute Anleitung unter
  `http://john.albin.net/git/convert-subversion-to-git`

Toolunterstützung

- Hosting von Repositories
- Entwicklungstools
- Build-, Release- und Projektverwaltung

- Dateisystem, Fileshare

- Dateisystem, Fileshare
- eigener Server z.B. gitolite, ssh-Server oder Apache mit WebDAV

- Dateisystem, Fileshare
- eigener Server z.B. gitolite, ssh-Server oder Apache mit WebDAV
- spezialisierte Git-Hoster wie Gitorious [7] oder GitHub [8]

- Dateisystem, Fileshare
- eigener Server z.B. gitolite, ssh-Server oder Apache mit WebDAV
- spezialisierte Git-Hoster wie Gitorious [7] oder GitHub [8]
- Entwicklungsplattformen wie SourceForge [9] oder Google Code [10]

IDEs und Editoren

- Eclipse – `http://www.eclipse.org/egit`

IDEs und Editoren

- Eclipse – `http://www.eclipse.org/egit` – Demo

IDEs und Editoren

- Eclipse – `http://www.eclipse.org/egit`
- IntelliJ IDEA – inklusive

IDEs und Editoren

- Eclipse – `http://www.eclipse.org/egit`
- IntelliJ IDEA – inklusive
- Netbeans – inklusive ab Netbeans 7.0

IDEs und Editoren

- Eclipse – `http://www.eclipse.org/egit`
- IntelliJ IDEA – inklusive
- Netbeans – inklusive ab Netbeans 7.0
- GNU Emacs – Basisupport inklusive ab 22.1, erweiterte Unterstützung mit Zusatzmodulen, Mode für Commit-Messages

IDEs und Editoren

- Eclipse – `http://www.eclipse.org/egit`
- IntelliJ IDEA – inklusive
- Netbeans – inklusive ab Netbeans 7.0
- GNU Emacs – Basisupport inklusive ab 22.1, erweiterte Unterstützung mit Zusatzmodulen, Mode für Commit-Messages
- VIM – diverse Wrapper, Syntax-Highlighting für Commit-Messages ...

Repository-Browser

- diverse Webfrontends – gitweb, cgit, ...

Repository-Browser

- diverse Webfrontends – gitweb, cgit, ...
- grafische History-Viewer – gitk, gitg, gitx, ...

Repository-Browser

- diverse Webfrontends – gitweb, cgit, ...
- grafische History-Viewer – gitk, gitg, gitx, ...

Commit-/Merge-Tools

- git-gui

Repository-Browser

- diverse Webfrontends – gitweb, cgit, ...
- grafische History-Viewer – gitk, gitg, gitx, ...

Commit-/Merge-Tools

- git-gui
- TortoiseGit, SmartGit

Repository-Browser

- diverse Webfrontends – gitweb, cgit, ...
- grafische History-Viewer – gitk, gitg, gitx, ...

Commit-/Merge-Tools

- git-gui
- TortoiseGit, SmartGit
- meld, diffuse, TortoiseMerge, ...

Maven – SCM-Provider

http://maven.apache.org/scm/git.html

```
<project xmlns="http://maven.apache.org/POM/4.0.0" ...>
  ...
  <scm>
    <connection>scm:git:http://projectserver/repo.git</connection>
    <developerConnection>scm:git:ssh://projectserver/git/repo.git</developerConnection>
      <url>http://projectserver/gitweb/?p=repo.git</url>
  </scm>
  ...
</project>
```

Maven – SCM-Provider

http://maven.apache.org/scm/git.html

```
<project xmlns="http://maven.apache.org/POM/4.0.0" ...>
  ...
  <scm>
    <connection>scm:git:http://projectserver/repo.git</connection>
    <developerConnection>scm:git:ssh://projectserver/git/repo.git</developerConnection>
      <url>http://projectserver/gitweb/?p=repo.git</url>
  </scm>
  ...
</project>
```

Ant – über Macros

http://tlrobinson.net/blog/2008/11/13/ant-tasks-for-git/

Continuous Integration

- Jenkins CI/ Hudson CI

Continuous Integration

- Jenkins CI/ Hudson CI – Demo

Continuous Integration

- Jenkins CI/ Hudson CI
- Cruise Control

Continuous Integration

- Jenkins CI/ Hudson CI
- Cruise Control
- Buildbot

Continuous Integration

- Jenkins CI/ Hudson CI
- Cruise Control
- Buildbot

Code-Reviews

- Gerrit [11]

Issue Tracking

- Atlassian JIRA (mit Git-Plugin)

Issue Tracking

- Atlassian JIRA (mit Git-Plugin) – Demo

Issue Tracking

- Atlassian JIRA (mit Git-Plugin)
- Trac (mit Plugin)

Issue Tracking

- Atlassian JIRA (mit Git-Plugin)
- Trac (mit Plugin)
- Bugzilla (mit GitZilla)

Issue Tracking

- Atlassian JIRA (mit Git-Plugin)
- Trac (mit Plugin)
- Bugzilla (mit GitZilla)
- Mantis (ab 1.2)

Issue Tracking

- Atlassian JIRA (mit Git-Plugin)
- Trac (mit Plugin)
- Bugzilla (mit GitZilla)
- Mantis (ab 1.2)
- Redmine

Workflows

- Grundlegendes
- Repositorystrukturen
- Branch-Workflows

- Git schreibt nichts vor

- Git schreibt nichts vor
- die Dokumentation liefert ein paar Anregungen [12]

- Git schreibt nichts vor
- die Dokumentation liefert ein paar Anregungen [12]
- im Internet gibt es eine Reihe von guten Tipps

- Git schreibt nichts vor
- die Dokumentation liefert ein paar Anregungen [12]
- im Internet gibt es eine Reihe von guten Tipps
  ⇨ Google: „git workflows“

- Git schreibt nichts vor
- die Dokumentation liefert ein paar Anregungen [12]
- im Internet gibt es eine Reihe von guten Tipps
  ⇨ Google: „git workflows“
- Git bietet an vielen Stellen Möglichkeiten mit Hooks einzugreifen [13]

- Git schreibt nichts vor
- die Dokumentation liefert ein paar Anregungen [12]
- im Internet gibt es eine Reihe von guten Tipps
  ⇨ Google: „git workflows“
- Git bietet an vielen Stellen Möglichkeiten mit Hooks einzugreifen [13]
- Tools wie Gerrit oder Gitolite helfen bei der Umsetzung von Workflows bzw. Durchsetzung von Richtlinien

Git Hooks

- clientseitig

Git Hooks

- clientseitig
  - Commit-Workflow – `pre-commit`, `prepare-commit-msg`, `commit-msg`, `post-commit`

Git Hooks

- clientseitig
  - Commit-Workflow – `pre-commit`, `prepare-commit-msg`, `commit-msg`, `post-commit`
  - E-Mail-Workflow – `applypatch-msg`, `pre-applypatch`, `post-applypatch`

Git Hooks

- clientseitig
  - Commit-Workflow – `pre-commit`, `prepare-commit-msg`, `commit-msg`, `post-commit`
  - E-Mail-Workflow – `applypatch-msg`, `pre-applypatch`, `post-applypatch`
  - sonstige – `pre-rebase`, `post-checkout`, `post-merge`

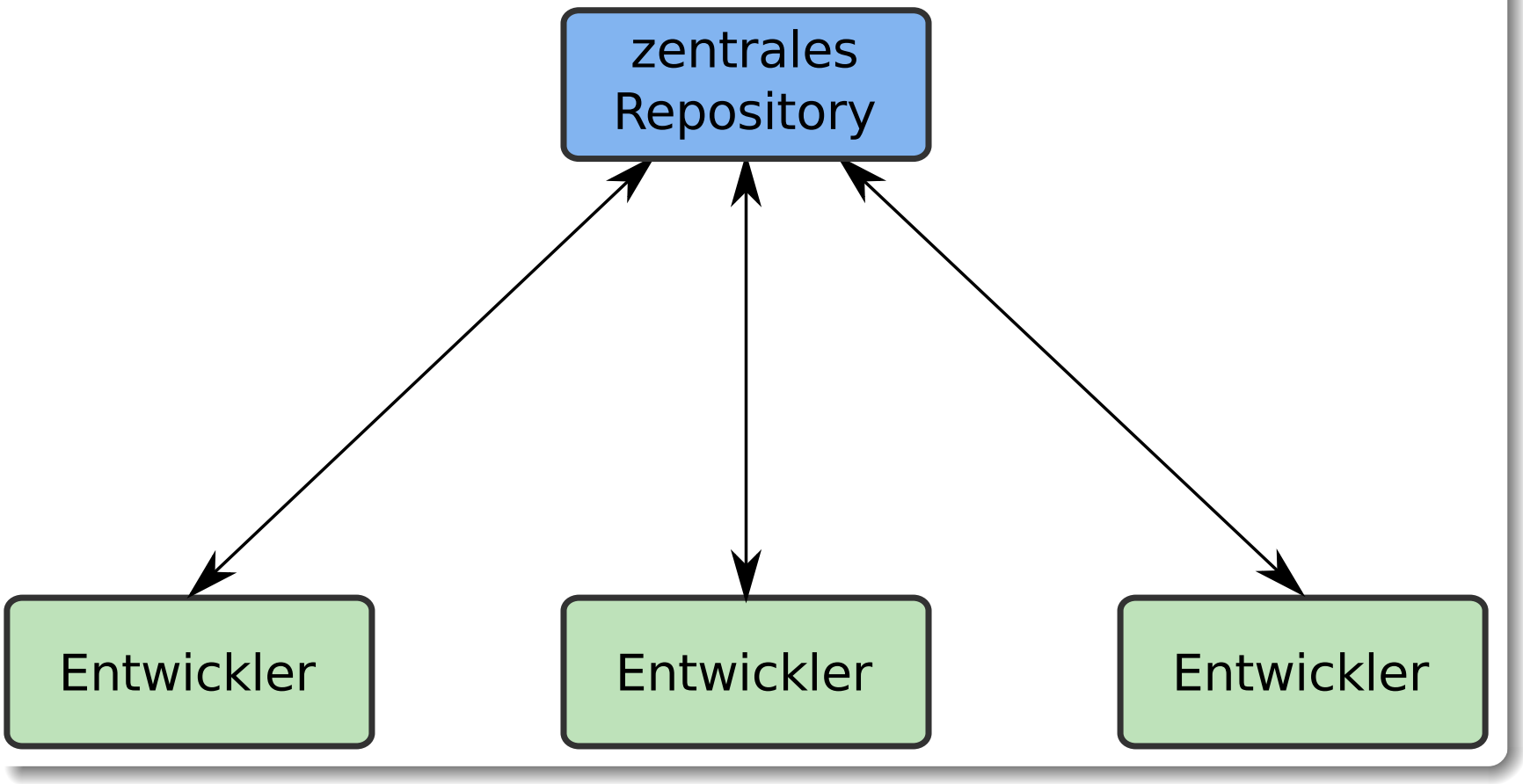
Git Hooks

- clientseitig
  - Commit-Workflow – `pre-commit`, `prepare-commit-msg`, `commit-msg`, `post-commit`
  - E-Mail-Workflow – `applypatch-msg`, `pre-applypatch`, `post-applypatch`
  - sonstige – `pre-rebase`, `post-checkout`, `post-merge`
- serverseitig

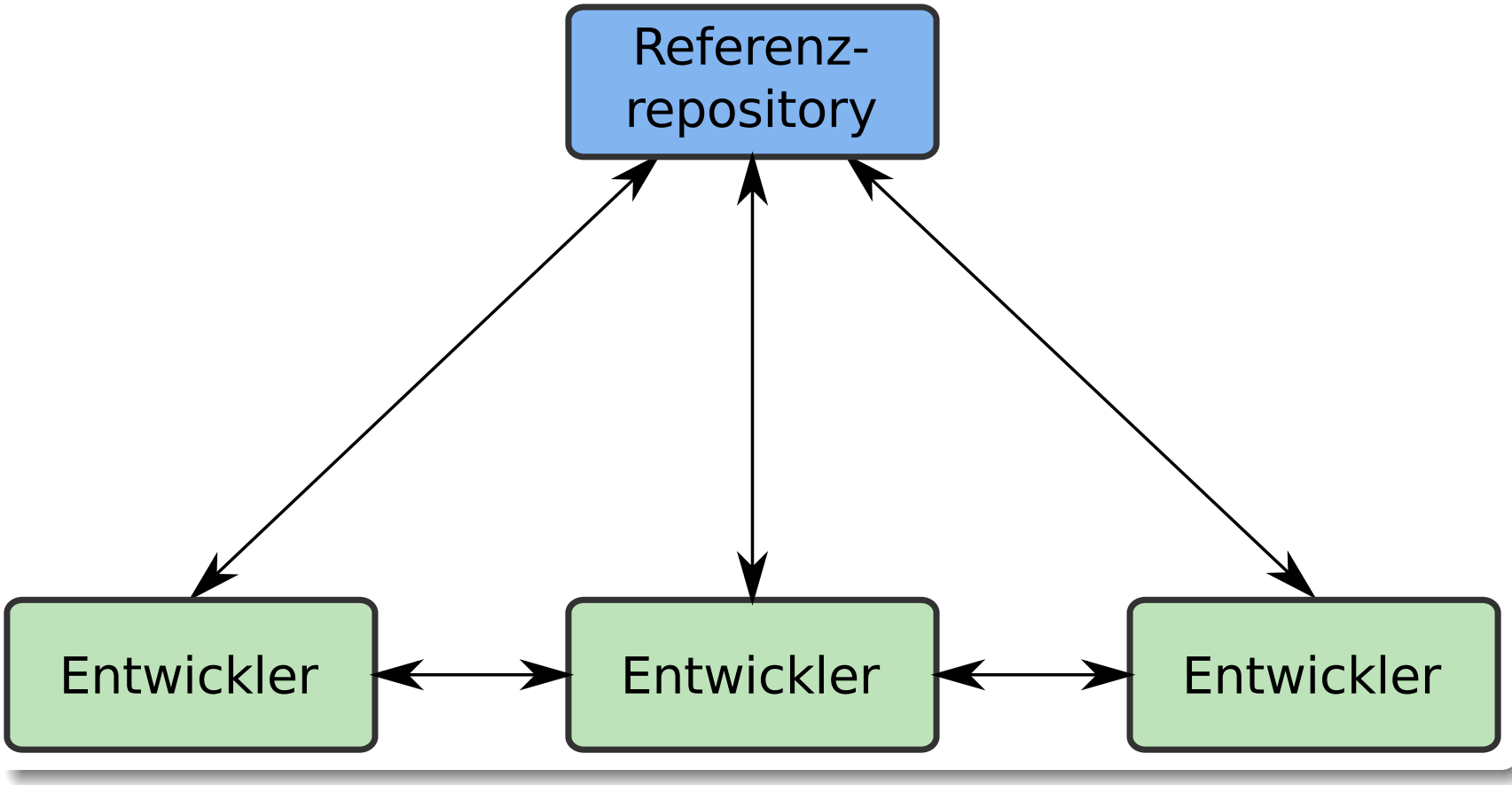
Git Hooks

- clientseitig
  - Commit-Workflow – `pre-commit`, `prepare-commit-msg`, `commit-msg`, `post-commit`
  - E-Mail-Workflow – `applypatch-msg`, `pre-applypatch`, `post-applypatch`
  - sonstige – `pre-rebase`, `post-checkout`, `post-merge`
- serverseitig
  - Eingehende Push-Requests – `pre-receive`, `post-receive`, `update`

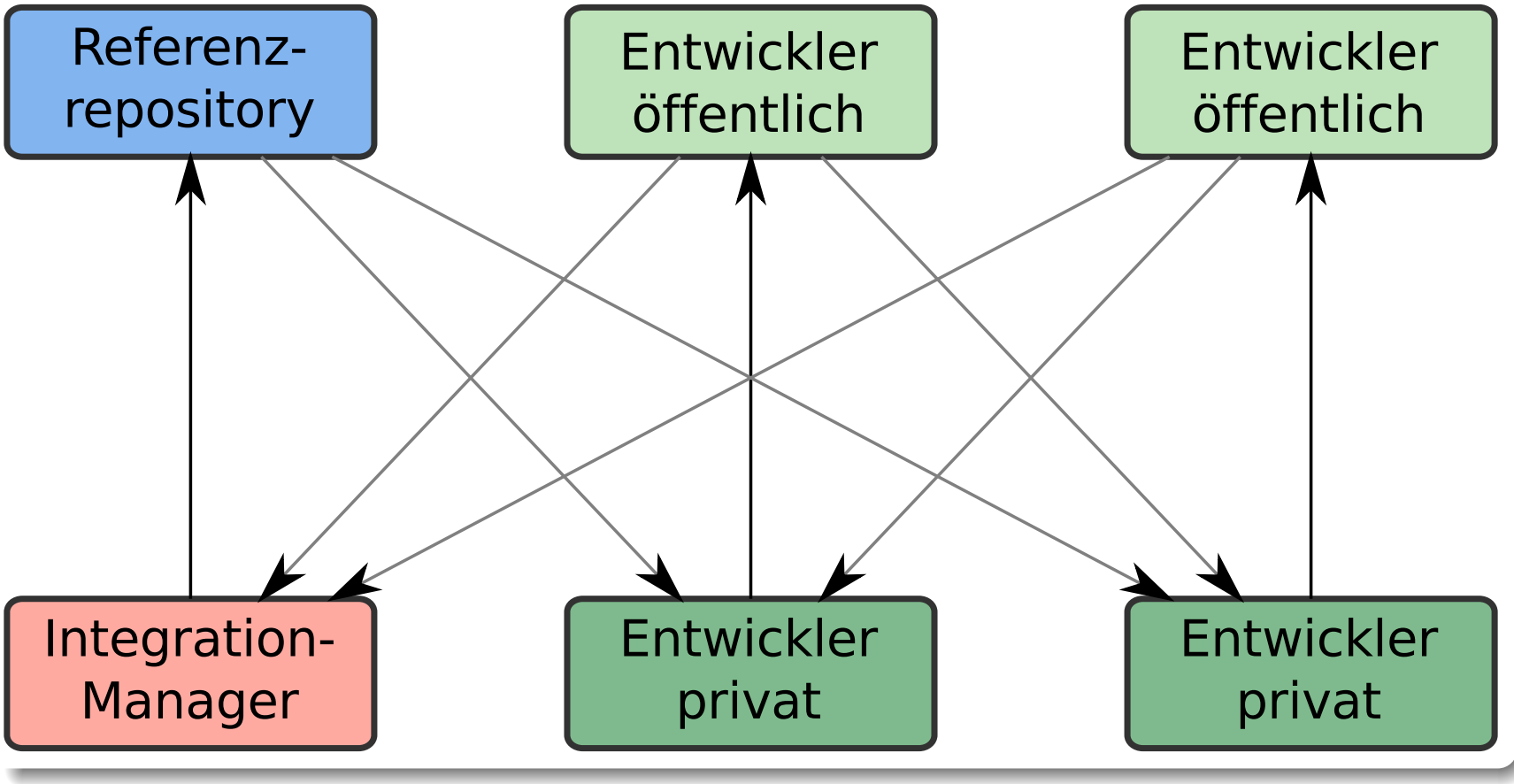
SVN-artig mit zentralem Repository



dezentral mit Referenzrepository

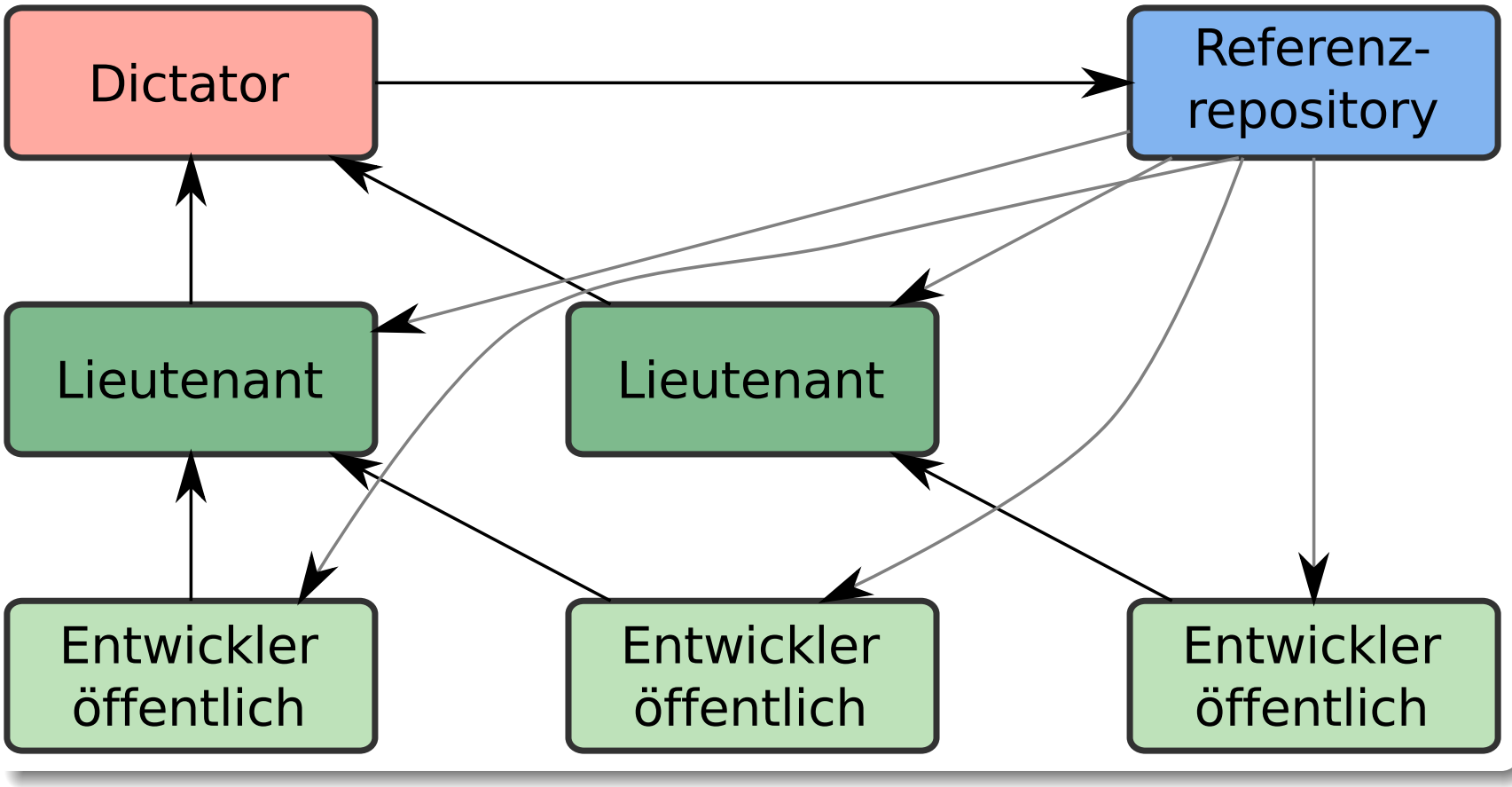


Integration-Manager



Dictator and Lieutenants[a]

[a] z.B. für Entwicklung des Linux-Kernels verwendet



A Git Workflow for Agile Teams

`http://reinh.com/blog/2009/03/02/a-git-workflow-for-agile-teams.html`

- regelmäßig pull von zentralem Repository in lokalen master

A Git Workflow for Agile Teams

`http://reinh.com/blog/2009/03/02/a-git-workflow-for-agile-teams.html`

- regelmäßig pull von zentralem Repository in lokalen master
- Einsatz von lokalen Featurebranches

A Git Workflow for Agile Teams

`http://reinh.com/blog/2009/03/02/a-git-workflow-for-agile-teams.html`

- regelmäßig pull von zentralem Repository in lokalen master
- Einsatz von lokalen Featurebranches
- Rebase der Featurebranches auf master

A Git Workflow for Agile Teams

`http://reinh.com/blog/2009/03/02/a-git-workflow-for-agile-teams.html`

- regelmäßig pull von zentralem Repository in lokalen master
- Einsatz von lokalen Featurebranches
- Rebase der Featurebranches auf master
- interaktives Rebase vor dem Merge

A Git Workflow for Agile Teams

`http://reinh.com/blog/2009/03/02/a-git-workflow-for-agile-teams.html`

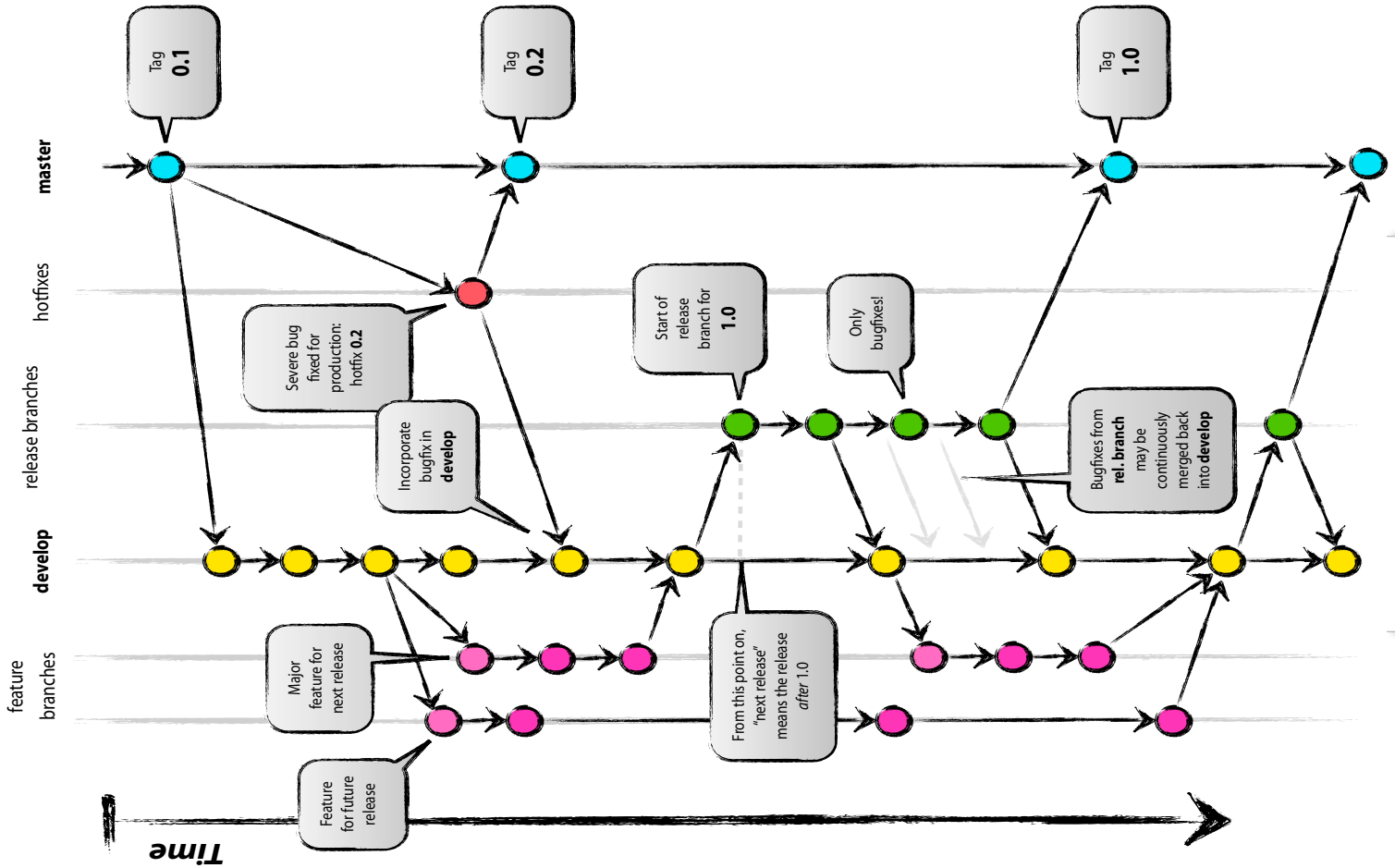
- regelmäßig pull von zentralem Repository in lokalen master
- Einsatz von lokalen Featurebranches
- Rebase der Featurebranches auf master
- interaktives Rebase vor dem Merge
- Merge des Featurebranches in Master

A Git Workflow for Agile Teams

`http://reinh.com/blog/2009/03/02/a-git-workflow-for-agile-teams.html`

- regelmäßig pull von zentralem Repository in lokalen master
- Einsatz von lokalen Featurebranches
- Rebase der Featurebranches auf master
- interaktives Rebase vor dem Merge
- Merge des Featurebranches in Master
- Push in zentrales Repository

A successful Git Branching Model [15]



A successful Git Branching Model [15]

- `master` für Lieferstände

A successful Git Branching Model [15]

- `master` für Lieferstände
- `develop` für Entwicklungsstände

A successful Git Branching Model [15]

- `master` für Lieferstände
- `develop` für Entwicklungsstände
- `release-*` für Releasevorbereitung

A successful Git Branching Model [15]

- `master` für Lieferstände
- `develop` für Entwicklungsstände
- `release-*` für Releasevorbereitung
- `hotfix-*` für Hotfixes

A successful Git Branching Model [15]

- `master` für Lieferstände
- `develop` für Entwicklungsstände
- `release-*` für Releasevorbereitung
- `hotfix-*` für Hotfixes
- Featurebranches

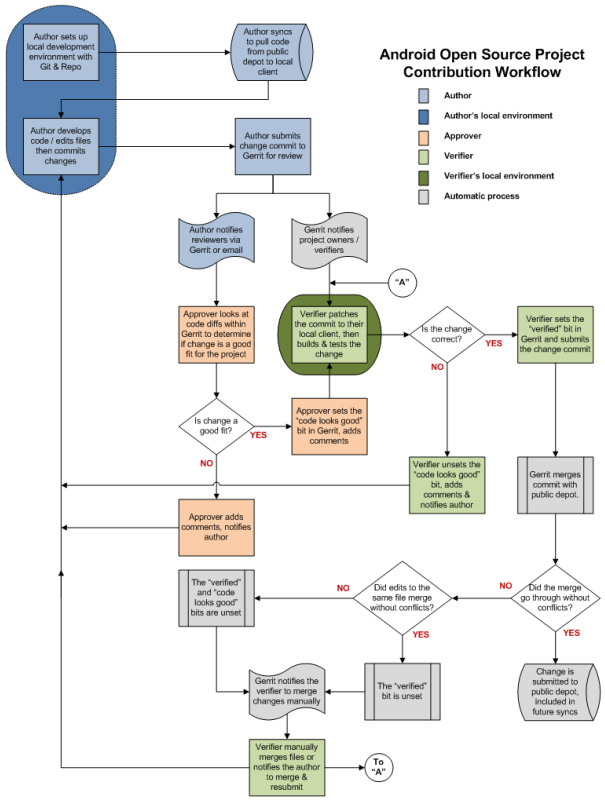
A successful Git Branching Model [15]

- `master` für Lieferstände
- `develop` für Entwicklungsstände
- `release-*` für Releasevorbereitung
- `hotfix-*` für Hotfixes
- Featurebranches

- Unterstützung durch git-flow [14] (Debian-Paket: git-flow)

Einbeziehung von CI-Tools und/oder Code-Reviews

`http://source.android.com/source/life-of-a-patch.html`



Blick über den Tellerrand

Blick über den Tellerrand

weiter interessante Anwendungsfälle für Git

- Verteiltes Bugtracking

Blick über den Tellerrand

weiter interessante Anwendungsfälle für Git

- Verteiltes Bugtracking
- Verwaltung von Metadaten zu großen Dateien

Blick über den Tellerrand

weiter interessante Anwendungsfälle für Git

- Verteiltes Bugtracking
- Verwaltung von Metadaten zu großen Dateien
- Wikis
- …`https://git.wiki.kernel.org/index.php/InterfacesFrontendsAndTools`

Anhang

`svn info`	`git config -l`
`svn status`	`git status`
`svn update`	`git pull`
`svn add`	`git add`
-	`git commit`
`svn commit`	`git push`
`svn diff`	`git diff`
`svn merge`	`git merge`
`svn copy`	`git branch` bzw. `git tag`
-	`git stash`
`svn merge -c <rev>`	`git cherry-pick`

Kontakt

- Jan Dittberner
- Communardo Software GmbH
- E-Mail: jan.dittberner@communardo.de
- `http://www.communardo.de/home/techblog/author/jdi`

[1] *Git project website*. URL: http://www.git-scm.com/.

[2] Scott Chacon. *Pro Git*. Apress, 2009. ISBN: 1430218339. URL: http://progit.org/book/.

[3] *Fedora Extra Packages for Enterprise Linux*. URL: http://fedoraproject.org/wiki/EPEL.

[4] *msysgit – Git for Windows*. URL: http://code.google.com/p/msysgit/.

[5] *Cygwin*. URL: http://www.cygwin.com/.

[6] *TortoiseGit*. URL: http://code.google.com/p/tortoisegit/.

[7] *Gitorious*. URL: http://gitorious.org/.

[8] *Github.com*. URL: https://github.com/.

[9] *SourceForge*. URL: http://www.sourceforge.net/.

[10] *Google Code*. URL: http://code.google.com/.

[11] *Gerrit Code Review*. URL: http://code.google.com/p/gerrit/.

[12] *Git Manual – gitworkflows (7)*.

[13] *Git Manual – githooks (5)*.

[14] Vincent Driessen. *gitflow*. URL: https://github.com/nvie/gitflow.

[15] Vincent Driessen. *A successful Git branching model*. 2010. URL: http://nvie.com/posts/a-successful-git-branching-model/.