Versionskontrolle mit `git`

David Kastrup

2. März 2008

Warum Sourcecodemanagement?

- Kooperation mit anderen Mitarbeitern

Warum Sourcecodemanagement?

- Kooperation mit anderen Mitarbeitern
- Zentrale Datenvorhaltung

Warum Sourcecodemanagement?

- Kooperation mit anderen Mitarbeitern
- Zentrale Datenvorhaltung
- Verfolgung und Dokumentation von Änderungen

Warum Dezentralisierung?

- Arbeit offline

Warum Dezentralisierung?

- Arbeit offline
- Vorstrukturierung von Änderungen

Warum Dezentralisierung?

- Arbeit offline
- Vorstrukturierung von Änderungen
- Schnelle Analyse der Historie

Warum Dezentralisierung?

- Arbeit offline
- Vorstrukturierung von Änderungen
- Schnelle Analyse der Historie
- Eigene Ideen und Änderungen weiterverfolgen

Abgrenzung zu anderen DVCS

- Schnell

Abgrenzung zu anderen DVCS

- Schnell
- Mäßig portierbar (Windows)

Abgrenzung zu anderen DVCS

- Schnell
- Mäßig portierbar (Windows)
- Leicht organisch gewachsen mit mehrfachen Redesigns (vgl. Linuxkernel)

Abgrenzung zu anderen DVCS

- Schnell
- Mäßig portierbar (Windows)
- Leicht organisch gewachsen mit mehrfachen Redesigns (vgl. Linuxkernel)
- Patch-basierter Workflow

Abgrenzung zu anderen DVCS

- Schnell
- Mäßig portierbar (Windows)
- Leicht organisch gewachsen mit mehrfachen Redesigns (vgl. Linuxkernel)
- Patch-basierter Workflow
- Mäßige Konsistenz in den Oberflächen

Abgrenzung zu anderen DVCS

- Schnell
- Mäßig portierbar (Windows)
- Leicht organisch gewachsen mit mehrfachen Redesigns (vgl. Linuxkernel)
- Patch-basierter Workflow
- Mäßige Konsistenz in den Oberflächen
- Quasi Nullaufwand für Start und Branches

Abgrenzung zu anderen DVCS

- Schnell
- Mäßig portierbar (Windows)
- Leicht organisch gewachsen mit mehrfachen Redesigns (vgl. Linuxkernel)
- Patch-basierter Workflow
- Mäßige Konsistenz in den Oberflächen
- Quasi Nullaufwand für Start und Branches
- Recht leicht, vollkommenes Chaos anzurichten

Abgrenzung zu anderen DVCS

- Schnell
- Mäßig portierbar (Windows)
- Leicht organisch gewachsen mit mehrfachen Redesigns (vgl. Linuxkernel)
- Patch-basierter Workflow
- Mäßige Konsistenz in den Oberflächen
- Quasi Nullaufwand für Start und Branches
- Recht leicht, vollkommenes Chaos anzurichten
- Recht leicht, das wieder ungeschehen machen zu können

Abgrenzung zu anderen DVCS

- Schnell
- Mäßig portierbar (Windows)
- Leicht organisch gewachsen mit mehrfachen Redesigns (vgl. Linuxkernel)
- Patch-basierter Workflow
- Mäßige Konsistenz in den Oberflächen
- Quasi Nullaufwand für Start und Branches
- Recht leicht, vollkommenes Chaos anzurichten
- Recht leicht, das wieder ungeschehen machen zu können
- Aufräumarbeiten an Patchserien normal und erwünscht

Workflow privater Änderungen: Merge vs. Rebase

Merge Zwei Versionen der Software werden zusammengeführt, Mergekonflikte automatisch/manuell aufgelöst: „am Ball bleiben“

Workflow privater Änderungen: Merge vs. Rebase

Merge Zwei Versionen der Software werden zusammengeführt, Mergekonflikte automatisch/manuell aufgelöst: „am Ball bleiben“

Rebase Ausgehend von einer gemeinsamen Ursprungsversion werden alle Patches des Entwicklungszweigs, die nicht schon verwendet wurden, auf die Ursprungsversion angewandt. „Getrennte Entwicklung mit späterer Zusammenführung“.

Workflow privater Änderungen: Merge vs. Rebase

Merge Zwei Versionen der Software werden zusammengeführt, Mergekonflikte automatisch/manuell aufgelöst: „am Ball bleiben“

Rebase Ausgehend von einer gemeinsamen Ursprungsversion werden alle Patches des Entwicklungszweigs, die nicht schon verwendet wurden, auf die Ursprungsversion angewandt. „Getrennte Entwicklung mit späterer Zusammenführung“.

Mischformen Bei wenig Überlappungen ist Merge zwischendurch unproblematisch. Nutzung von Mergehistory hilfreich.

Wann was?

Merge Parallele Entwicklungszweige mit signifikanten Änderungen vorzugsweise an wenig überlappenden Stellen, schneller Abgleich paralleler Arbeiten.

Wann was?

Merge Parallele Entwicklungszweige mit signifikanten Änderungen vorzugsweise an wenig überlappenden Stellen, schneller Abgleich paralleler Arbeiten.

Rebase

- Private Patches, die (noch) nicht für andere interessant sind
- Entwicklung von Features als “Patch set”
- Unsystematisch: Aufräumaktionen irrelevanter Halbentwicklungen
- Aber: prinzipiell Workflow, der *aufgeräumte* Patch Sets bevorzugt, da Mergekonfliktauflösung potentiell für jeden Patch anfällt.

Typischer Workflow insgesamt

- Private Entwicklung

Typischer Workflow insgesamt

- Private Entwicklung
- Regelmäßiger Merge/Pull der Hauptentwicklung

Typischer Workflow insgesamt

- Private Entwicklung
- Regelmäßiger Merge/Pull der Hauptentwicklung
- Bei Bedarf für Ordnung (wegen Submit von Patches oder persönlicher Ästhetik) Rebase

Implementationsdetails

- Viele, viele bunte Shellscripts

Implementationsdetails

- Viele, viele bunte Shellscripts
- Erfordert POSIX-Shell und C

Implementationsdetails

- Viele, viele bunte Shellscripts
- Erfordert POSIX-Shell und C
- Einige Utilities in Perl, Tcl/Tk

Implementationsdetails

- Viele, viele bunte Shellscripts
- Erfordert POSIX-Shell und C
- Einige Utilities in Perl, Tcl/Tk
- Portierung auf Windows (per MSYS) mittlerweile verfügbar

Implementationsdetails

- Viele, viele bunte Shellscripts
- Erfordert POSIX-Shell und C
- Einige Utilities in Perl, Tcl/Tk
- Portierung auf Windows (per MSYS) mittlerweile verfügbar
- Effiziente Dateisysteme wichtig

Implementationsdetails

- Viele, viele bunte Shellscripts
- Erfordert POSIX-Shell und C
- Einige Utilities in Perl, Tcl/Tk
- Portierung auf Windows (per MSYS) mittlerweile verfügbar
- Effiziente Dateisysteme wichtig
- Aufteilung Porcelaine/Plumbing

Implementationsdetails

- Viele, viele bunte Shellscripts
- Erfordert POSIX-Shell und C
- Einige Utilities in Perl, Tcl/Tk
- Portierung auf Windows (per MSYS) mittlerweile verfügbar
- Effiziente Dateisysteme wichtig
- Aufteilung Porcelaine/Plumbing
- Migration `sh` $\mapsto$ `C` ingange

Praxisdetails

- Sehr kompakte Archive

Praxisdetails

- Sehr kompakte Archive
- Schnell (erfordert effizientes Dateisystem)

Praxisdetails

- Sehr kompakte Archive
- Schnell (erfordert effizientes Dateisystem)
- Interaktionstools ohne übergreifendes Design

Geringe Hemmschwellen

- Anlegen eines `git`-Projektes: `git init` im fraglichen Verzeichnis

Geringe Hemmschwellen

- Anlegen eines `git`-Projektes: `git init` im fraglichen Verzeichnis
- Anlegen eines Branches: `git branch` *`name`*, auch falls noch Commits ausstehen

Geringe Hemmschwellen

- Anlegen eines `git`-Projektes: `git init` im fraglichen Verzeichnis
- Anlegen eines Branches: `git branch` *`name`*, auch falls noch Commits ausstehen
- Lokale Änderungen kurzzeitig verstauen per `git stash`

Guerilla-Entwicklung mit git

Keine Kooperation von anderen Projektteilnehmern erforderlich.

- Subversion-Anbindung in zwei Richtungen per `git-svn`

Guerilla-Entwicklung mit git

Keine Kooperation von anderen Projektteilnehmern erforderlich.

- Subversion-Anbindung in zwei Richtungen per `git-svn`
- Anbindung an CVS über verschiedene Einzeltools zum Komplettimport oder Commit einzelner Patches

Guerilla-Entwicklung mit git

Keine Kooperation von anderen Projektteilnehmern erforderlich.

- Subversion-Anbindung in zwei Richtungen per `git-svn`
- Anbindung an CVS über verschiedene Einzeltools zum Komplettimport oder Commit einzelner Patches
- Import von Arch

Guerilla-Entwicklung mit git

Keine Kooperation von anderen Projektteilnehmern erforderlich.

- Subversion-Anbindung in zwei Richtungen per `git-svn`
- Anbindung an CVS über verschiedene Einzeltools zum Komplettimport oder Commit einzelner Patches
- Import von Arch
- Import von Bitkeeper

git-svn

- Rebase-basierter Workflow

git-svn

- Rebase-basierter Workflow
- Geschrieben in Perl

git-svn

- Rebase-basierter Workflow
- Geschrieben in Perl
- Langsamer Import einer Gesamthistorie, lokaler Mirror des Archives sinnvoll

git-svn

- Rebase-basierter Workflow
- Geschrieben in Perl
- Langsamer Import einer Gesamthistorie, lokaler Mirror des Archives sinnvoll
- Selbst Readonly sinnvoll wegen guten Visualisierungstools und zu Experimenten

Community, Entwicklungsmodell

- Entstanden durch „Bitkeeper-Entzug“ in wenigen Wochen, hauptsächlich von Linus Torvalds konzipiert und zunächst implementiert

Community, Entwicklungsmodell

- Entstanden durch „Bitkeeper-Entzug“ in wenigen Wochen, hauptsächlich von Linus Torvalds konzipiert und zunächst implementiert
- Maintainer ist Junio Hamano aus Japan

Community, Entwicklungsmodell

- Entstanden durch „Bitkeeper-Entzug“ in wenigen Wochen, hauptsächlich von Linus Torvalds konzipiert und zunächst implementiert
- Maintainer ist Junio Hamano aus Japan
- Mailingliste hochkompetent mit einigen „nononsense“-Fetischisten, dort teilweise rüder Ton.

Community, Entwicklungsmodell

- Entstanden durch „Bitkeeper-Entzug“ in wenigen Wochen, hauptsächlich von Linus Torvalds konzipiert und zunächst implementiert
- Maintainer ist Junio Hamano aus Japan
- Mailingliste hochkompetent mit einigen „nononsense“-Fetischisten, dort teilweise rüder Ton.
- Lizenz GPLv2, Patches werden nach Diskussion üblicherweise von Mailingliste in Hauptzweig importiert oder von Submaintainern gezogen.