The Git Source Code Management System

- git @ work

or:

a little bit of git

-
-
- Mark Struberg
- Vienna, 2008
-

Was ist ein SCM

SCM steht hier für **Source Code Management** und bringt folgende Vorteile

- **Checkin**: Organisierter gemeinsamer Zugriff auf Sourcen. (vs derjenige der als Letzter speichert hat gewonnen)
- **History**: Genaue Nachvollziehbarkeit wann und wer gewisse Zeilen hinzugefügt/geändert/gelöscht hat. Man hat also nicht nur den Letztstand sondern die komplette Historie des Sourcecodes verfügbar.
- **Branches**: Es ist möglich, mehrere Versionen des Sources zu verwalten. zB feature branches und maintainance branches

Was ist ein SCM (cntd)

- **Versioning**: Möglichkeit, den Zustand eines bestimmten Zeitpunktes später wieder genau zu rekonstruieren.
- **Tagging**: Möglichkeit den Zustand zu einem bestimmten Zeitpunkt 'einzufrieren'
- **Annotations**: Nachvollziehbarkeit aus welchem Grund gewisse Änderungen hinzugefügt wurden.
- Jede **Änderung** kann mit einem **Kommentar** versehen werden.

Source Code Management vs Software Configuration Management

- **Source Code Management** ist nur ein Teilbereich von Software Configuration Management.
- **Software Configuration Management** beinhaltet einen viel breiteren Ansatz und enthält als Zusammenfassung von diversen Hardware-, Software und Dienstleistungseinheiten zB folgende zusätzliche Aspekte:

Software Configuration Management

- **Change Control**: Wie werden Änderungswünsche am System geplant und umgesetzt
- **Configuration Control**: Welche verschiedenen Ausprägungen meines Produktes gibt es
- **Feature Planing**: Wie werden neue Anforderungen an das Produkt geplant und umgesetzt
- **Release Planing**: Wann werden neue Releases freigegeben, was passiert mit den alten Releases
- **Resource Planing**: Wieviele Programmierer, Testpersonen, Projektleiter, etc benötige ich
- **Documentation, Training & Support**: Wie bringe ich das Wissen über das System zu demjenigen der es schlußendlich benötigt

generelle Kategorisierung von SCMs

Es gibt ein paar typische Merkmale die sich im Laufe der Zeit entwickelt haben:

- getrennter Zugriff auf **zentrale Datenablage**
- **Client-Server** basierte SCMs
- **verteilte SCM**s ohne technisches 'Mastersystem'
- **file locking** vs **conflict merging**

Vergleich unterschiedlicher SCMs

- Patches
- CVS
- SVN
- SourceSafe
- Mercurial
- BitKeeper

SCM Vergleich: CVS

- CVS steht für Concurrent Versions System
- CVS bestand **ursprünglich** (1989) aus einigen **perl und shell scrips**, in der Zwischenzeit ist es aber vollständig in C implementiert.
- CVS ist ein **strikt client-server basiert**es System
- only conflict merging
- sehr stabil
- sehr gutes Toolset
- läuft auf allen Betriebssystemen
- halbwegs flott

SCM Vergleich: **Subversion (SVN)**

- SVN ist das inoffizielle Nachfolgeprojekt von CVS und wurde 2000 offiziell vorgestellt.
- commits sind 'true atomic' functions
- versionierte Datei-Metadaten. Dadurch sind auch file deletes und moves historisiert verfügbar.
- Subversion wurde in der Version 1.4 um Ansätze einer dezentralen Lösung erweitert

SCM Vergleich: Visual SourceSafe

- Visual SourceSafe ist das Microsoft eigene SCM.
- rein filebasiert, der Zugriff erfolgt über einen shared folder
- verliert ab einer gewissen Größe sporadisch Dateien
- neigt dazu, ab einer gewissen Zeit kaputt zu werden.
- branching is a real nightmare
- gute Integration in VisualStudio
- kann standalone verwendet werden
- läuft nur unter MS Windows
- lizenzpflichtig

SCM Vergleich: Mercurial

- dezentrales System
- relativ langsames Protokoll
- funktioniert gut
- gutes Toolset
- in Python geschrieben

SCM Vergleich: BitKeeper

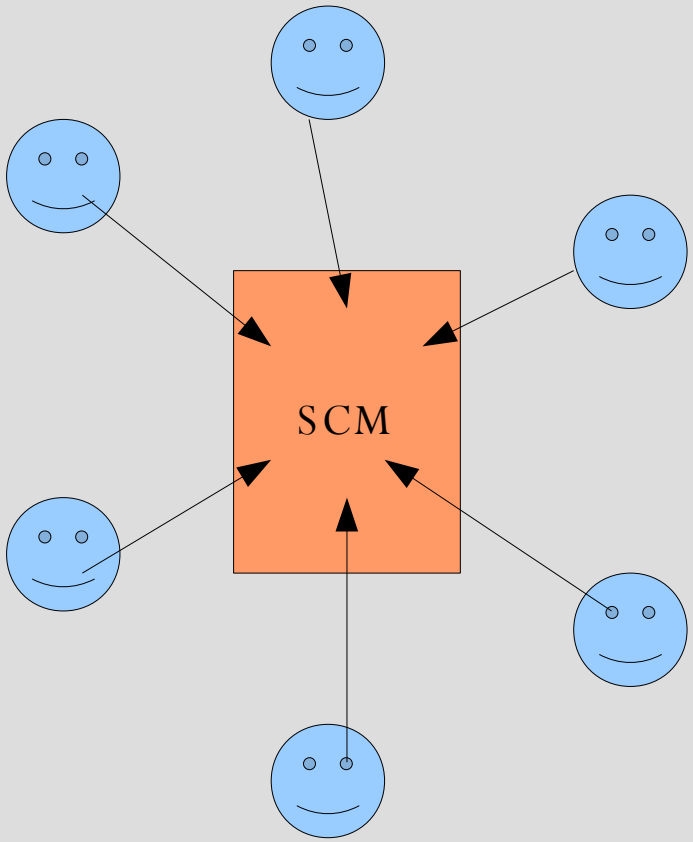
- dezentrales System mit zentraler Organisation
- Wurde früher zur Verwaltung des Linux Kernels verwendet
- Kommerzielles Produkt
- gratis für nonkommerziellen Einsatz
- Darf nicht reverse-engineered werden

Entwickeln mit Git

- Git gehört zur neuen Generation verteilter SCMs und erfordert eine leicht geänderte Arbeitsweise im Vergleich mit Client-Server Systemen.
- Linus Torvalds entwickelte git im April 2005, nachdem es einige Diskussionen über die Verwendung von BitKeeper zur Verwaltung des Linux kernels gab.
- Junio C. Hamano ist fast seit Anfang für die maintainance zuständig

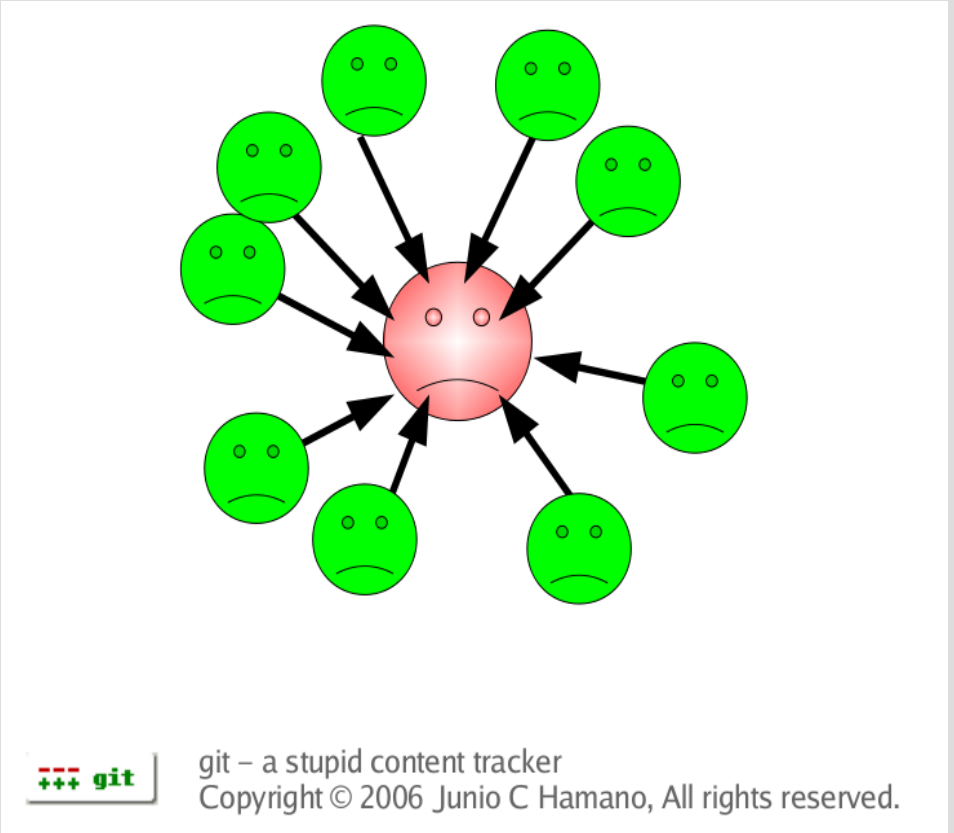
Topologie klassischer SCMs

- Client-Server Architektur
- Alle verwenden das selbe Repository
- Jeder Entwickler hat Schreibrechte
- Jeder checked seine Änderungen direkt selbst ein – kein vorgelagertes Review.



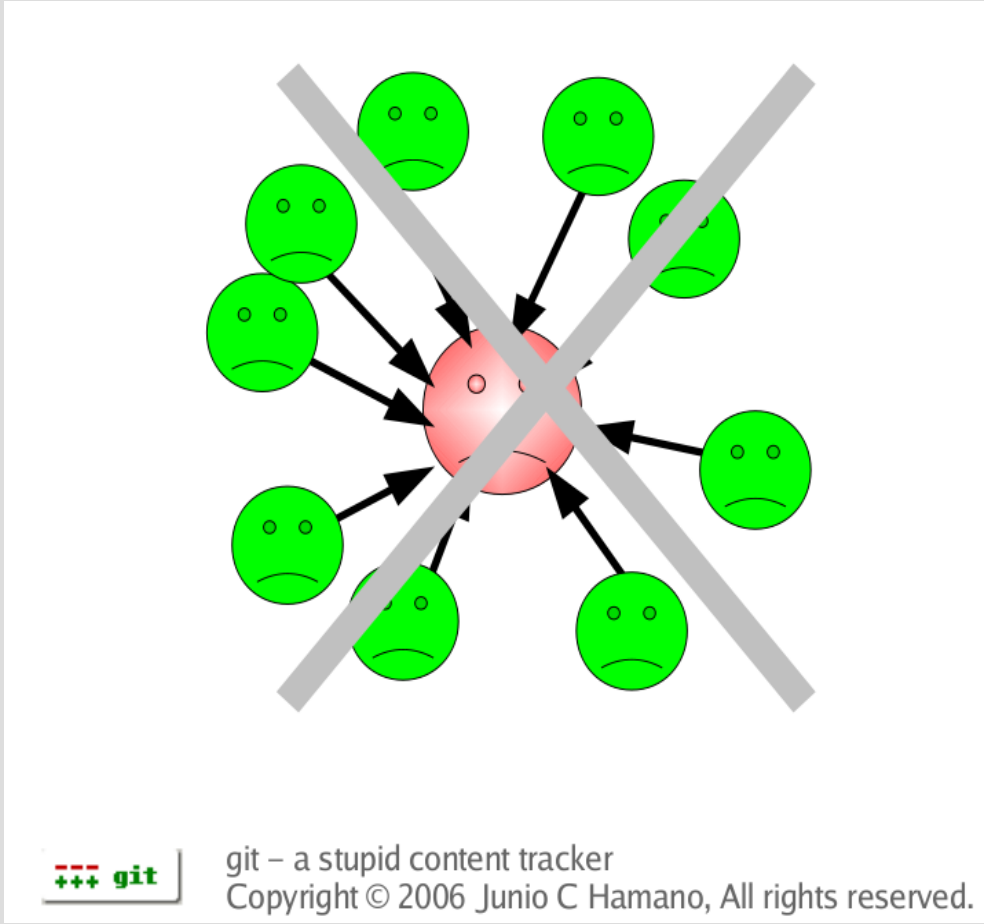
Linux kernel Verwaltung - old style

- Linus als einziger "Committer"
- Alle senden Patches an Linus
- Linus merged alle Patches
- Linus gibt die Änderungen frei

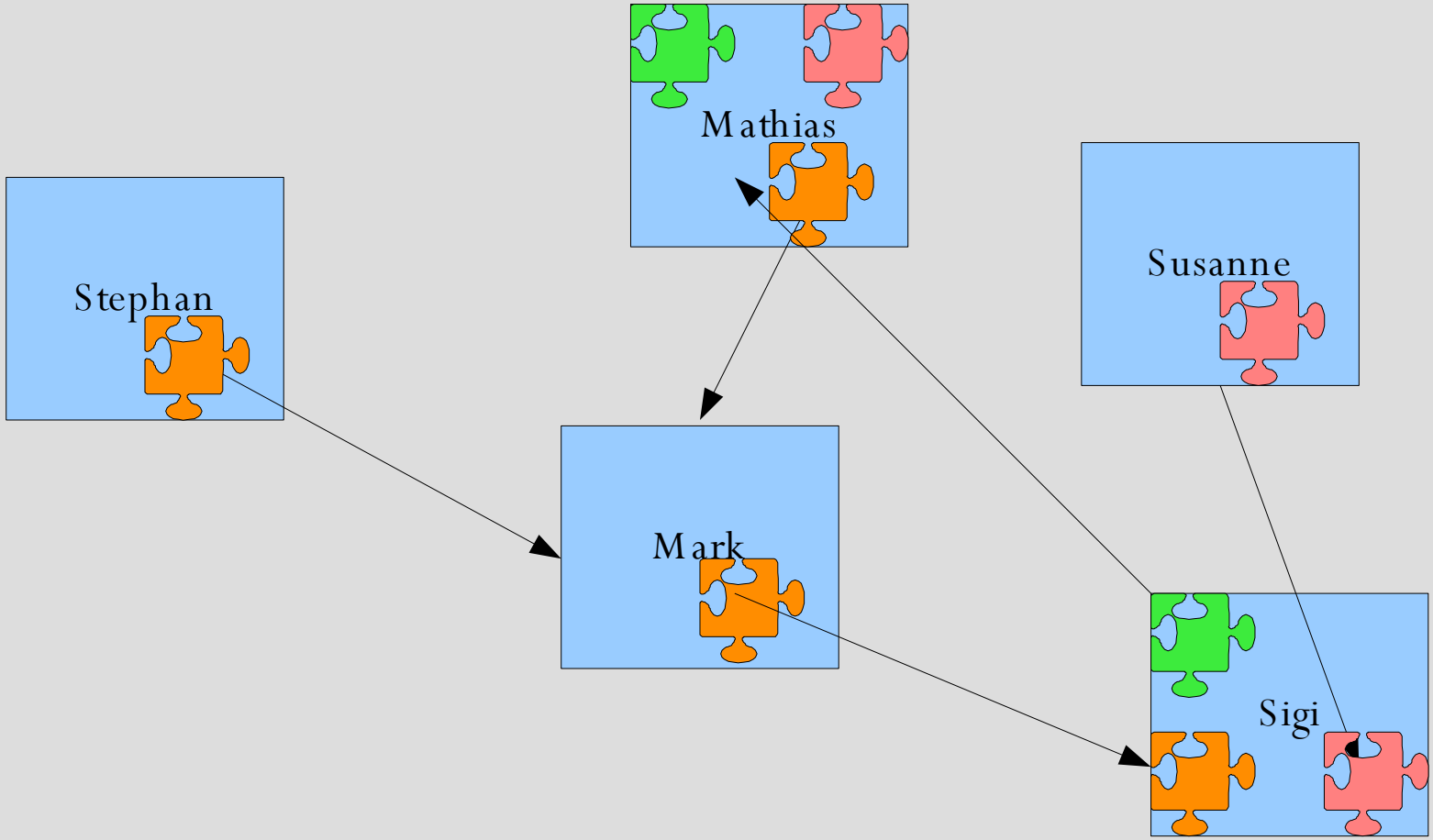


Probleme klassischer SCMs

- Klassische Client-Server SCMs erfordern eine strikt hierarchische Organisation
- Wenn der Server steht, dann geht fast gar nix
- offline arbeiten oft schwierig bis unmöglich
- In großen Projekten oft Probleme mit Rechtevergabe



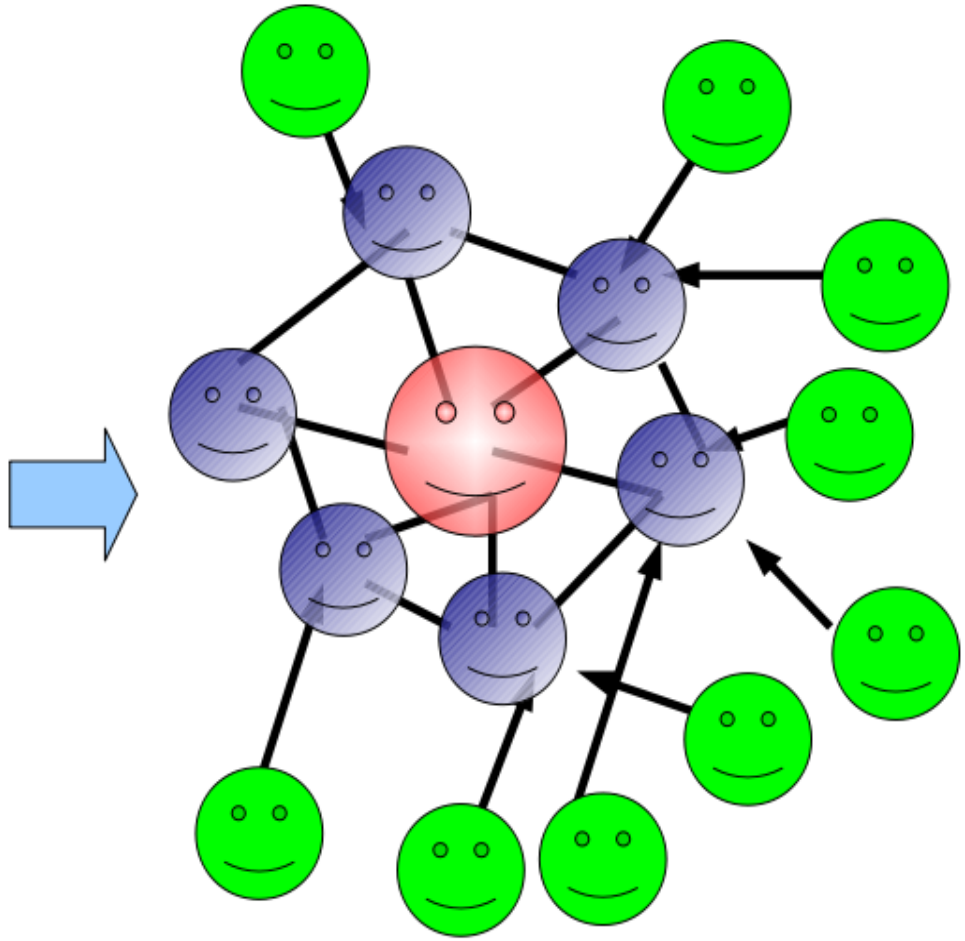
Topologie dezentraler SCMs



Verwaltung von Linux mit Git

- *Not directly working with 100+ people.*
- *Network of "Fairly central person" and "Trusted Lieutenants".*
- *Not a hierarchy.*

This happened when "the SCM that came before git" was adopted by the kernel project.



git – a stupid content tracker

Network of Trust

- **Jeder** schreibt nur in sein **eigenes Repository**
- Dadurch **keine Berechtigungsprobleme**
- **Gute** Änderungen werden aktiv '**gepulled**'
- **Author** einer Änderung bleibt **erhalten**
- Es gibt kein technisches Mastersystem
- Selbstorganisation durch **Qualität**
- Einzelne Änderungen werden mehrfach gesichtet

Git Repository

- Pures **content-tracking** (kein file-tracking)
- Liste von Änderungen (diffs)
- Git sammelt **Änderungen innerhalb** eines **Baum**s
- Direkter Azyklischer Graph (**DAG**) von Änderungen
- Jede Änderung per **SHA1** hash eindeutig
- Dadurch implizite **Security**
- Zusätzlich **GPG** Signatur möglich
- Möglichkeit **verschachtelte Module** anzulegen
- Alle Information unter .git
- Unterstützt **viele** Übertragungs**protokolle**

Schwächen von Git

- Optimiert für gesamte **tree**s und nicht einzelne files
- **Annotation** ist relativ **teuer**
- Git verwaltet **keine Berechtigungen**
- Git verwaltet **keine** Datei **Metadaten**

Übertragungsprotokolle

- **file** – Zugriff auf lokale Repositories
- **ssh** – Authentizierung über Secure Socket Layer
- **http** – public internet. Schreiben per webdav
- **git** – natives Protokoll. Extrem schnell.
- Auch **gemischter Betrieb möglich**. ZB lesen per http, schreiben per ssh.

Git Befehle allgemein

- **git-befehl**
- oder
- **git befehl**
- Git Befehle arbeiten meist gegen den lokalen 'Index'
- **HEAD** zeigt immer auf die aktuelle Arbeit

Arbeiten mit lokalem Git Repo

- **git-init**
- **git-add**, **git-rm** von durchgeführten Änderungen
- **git-commit** um die Änderungen in das lokale Repository zu übernehmen
- optionales **git-push** auf ein 'public' Repository

Clonen eines bestehenden Repo

- **git-clone**
- **git-status**
- **git-add**, **git-rm** von durchgeführten Änderungen
- **git-commit** um die Änderungen in das lokale Repository zu übernehmen
- **git-push** um die Änderungen im lokalen Repository auf ein anderes zu verteilen.

Arbeiten mit branches

- Git vereinfacht das **branching**
- vor allem aber das **mergen**
- **maintainance**-branch
- **feature**-branch

Git branch Befehle

- **git-branch** [branchname] legt neuen branch an
- **git-checkout** [branchname] holt branches aus dem lokalen Repo
- **git-rebase** ermöglicht 'fast forward' von patches
- **git-tag** erzeugt einen Namen für einen SHA1 commit
- **git-checkout tag** [tagname] holt einen named tag aus dem lokalen Repo
- **git-cherry-pick** [SHA1] wendet einen einzelnen commit an

Mergen mit Git

- git verwendet einen **3-way merge**
- **git-merge** [branch1] [branch2]
- **git-pull** [remotebranch]
- Vorteil der **diff** Verwaltung gegenüber klassischen SCMs: einfacher merge von branches
- **git-bisect** zur Suche von Fehlern

Graphische Tools

- **gitk** - ist eine in perl/tcl geschriebene GUI zum browsen eines Repositories
- **git-gui** - ermöglicht graphisches git-status, git-add, git-commit, etc
- **egit** – rein Java basierte Implementierung für zB Eclipse und IDEA Einbindung

Weitere wichtige Git Befehle

- git-diff - zeige Änderungen zwischen Versionen
- git-log - zeigt Commits an
- git-status - zeigt aktuelle Änderungen
- git-push - verteilt commits an andere Repos
- git-reset - setzt den Index wieder zurück
- git-revert - erstellt und commited einen 'undo patch'
- git-ls-files - listet Dateien im Index und Tree
- git-ls-tree - zeigt den Inhalt eines Tree Objektes
- git-repack - packt mehrere diffs in Sammelobjekte
- git-gc - löscht unbenutzte Blobs (garbage collect)

My private Git

- Auch wenn im Firmen- / **Projektumfeld** ein **anderes SCM** verwendet wird, kann eine lokale Verwendung von Git durchaus sinnvoll sein.
- Vorteil bei **feature branch**es.
- Vorteil wenn **Kleingruppen** zusammenarbeiten müssen ohne das Hauptrepo mit Zwischenresultaten zuzumüllen.

Git „on top“

- Erstellen eines lokalen Git Repositories zur effizienten Verwaltung von feature branches, ohne diese in CVS oder SVN anlegen zu müssen.
- **git-init** neben .svn oder .cvs
- **.gitignore** und **.cvsignore** anpassen
- **git-branch** feature1
- **git-checkout** feature1
- **git-add**, **git-commit**
- zwischenzeitlich bugs fixen-> **git-checkout master**
- danach wieder **git-checkout feature1**
- abschließendes **checkin in CVS** oder SVN
- updaten des master branches im lokalen git

SCM Connectivity

- git-svn bridge
- git-cvsimport, git-cvsexportcommit
- git-cvsserver

Aufsetzen eines Git Servers

- Das Aufsetzen eines Git Servers ist stark davon **abhängig**, welches **Übertragungsprotokoll** man verwenden will.
- Allgemein empfiehlt sich die Verwendung von **Linux** als **Betriebssystem**, da hier 'Überraschungen' sehr unwahrscheinlich sind.

Git over http (Apache httpd)

- `$> mkdir /var/www/html/git` Verzeichnis angelegt
- erzeuge **empty git repo**sitory

```
$> export GIT_DIR=maven-scm-providers-git.git
$> git-init –shared=group
$> chown -R apache:apache *
$> vi maven-scm-providers-git.git/description
```

- serverinfo automatisieren. Bewirkt, daß die meta-info für den html download aktualisiert wird

```
$> git-update-server-info
$> chmod a+x maven-scm-providers-git.git/hooks/post-update
```

- **git-web** oder
- **git-php** http://code.google.com/p/git-php/

git-shell und ssh

- Anlegen eines users
  ```
  $> useradd myuser
  ```
- Git-shell als shell einrichten
  ```
  $> vi /etc/passwd
  myuser:x:500:500::/home/myuser:/usr/bin/git-shell
  ```
- `$> passwd myuser`
- `git-push ssh://myuser@myserver.net/var/www/html/git/maven-scm-providers-git.git/ master`

Legal aspects

- Git selbst wurde unter GPLv2 veröffentlicht
- egit teilweise unter GPL, teilweise LGPL
- maven-scm-providers-git unter ASL

Links

- http://www.kernel.org/pub/software/scm/git/docs/
  offizielle Git Dokumentation
- http://git.or.cz/
  Git wiki
- http://members.cox.net/junkio/200607-ols.pdf
  Vortrag von Junio C Hamano
- http://youtube.com/watch?v=4XpnKHJAok8
  Linus on Git
- http://video.google.com/videoplay?docid=-3999952944619244
  Randal Schwartz about git

Q&A