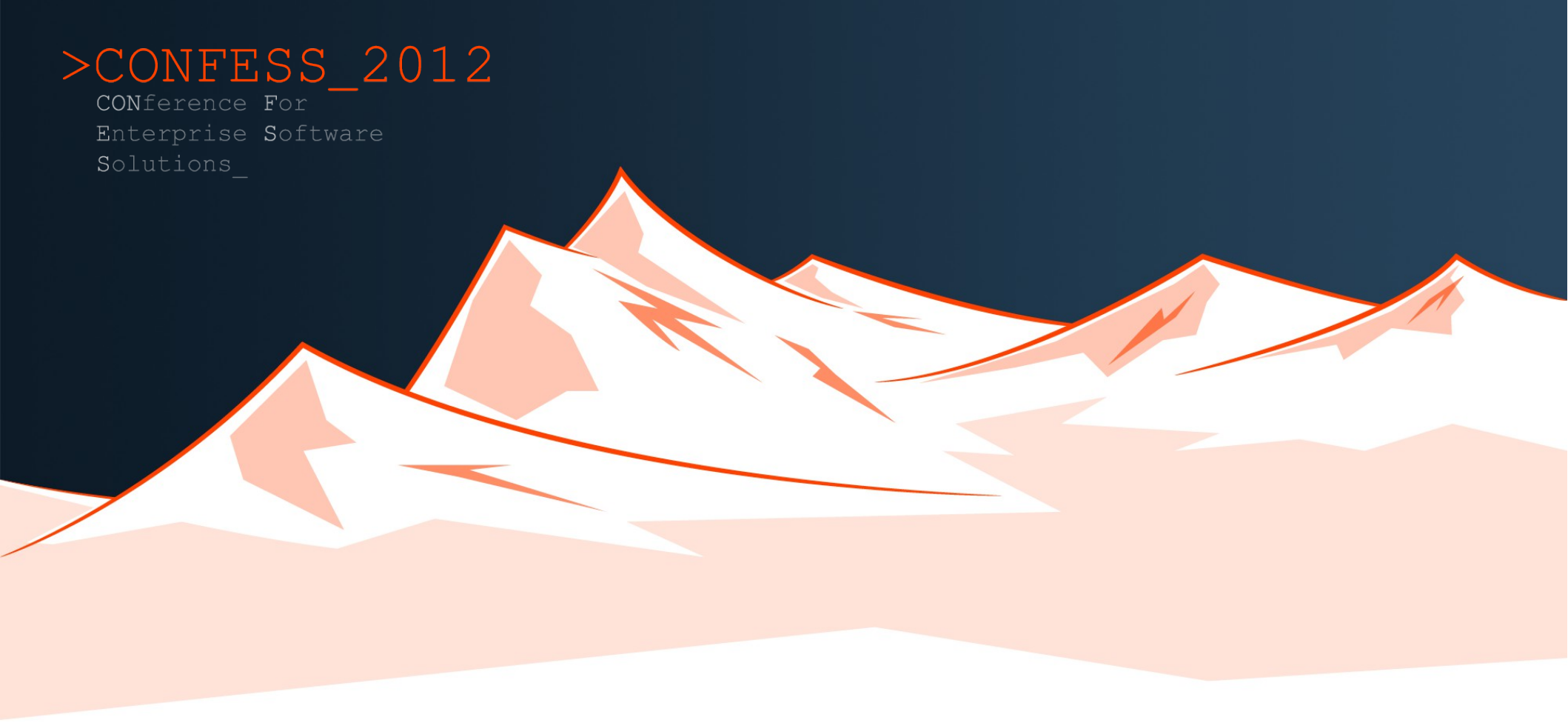


GIT for companies

Mark Struberg,
INSO TU Vienna

What is an SCM

SCM stands for **Source Code Management**

- **Checkin**: organized common access to sources
- **History**: who did commit which code at what time. This creates the full history of all the source code.
- **Versioning**: reconstruct an exact original state at a later time

What is an SCM (cntd)

- **Branches**: Maintaining multiple versions of the sources. E.g. feature branches and maintainance branches
- **Tagging**: 'freeze' a state and give it a special 'name'
- **Comments**: each change can have a comment which describes the reason for the change.
- **Annotations**: who change which line in a file?

Source Code Management vs Software Configuration Management

- **Source Code Management** is only a small aspect of Software Configuration Management.
- **Software Configuration Management** is a much broader approach containing many hardware, software and business aspects, like:

Software Configuration Management

- **Change Control**: handles change requirements
- **Configuration Control**: managing different derivations of a product
- **Feature Planing**: how to set new features into production
- **Release Planing**: what happens with old releases, and how to update to new releases
- **Resource Planing**: planing your programmers, testing personal, managers, etc
- **Documentation, Training & Support**: bring the knowledge to the one who needs it: the customer

general SCM categories

there are a few typical features which distinguish various SCMs:

- **Daemon** based SCMs vs F**ile** based
- **Distributed SCM**s without a technical 'master' system vs **Client-Server** systems.
- **file locking** vs **conflict merging**

Overview different SCM systems

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

SCM Comparison: Patches

- shipping diff patches on the mailing list
- always needs to get applied in the right order
- easy to historize
- good visibility
- hard to maintain manually

SCM Comparison: CVS

- CVS stands for Concurrent Versions System
- CVS originally (1989) consisted of a few **perl und shell scrips**, now fully implemented in **C**.
- CVS is **strctly client-server based**
- only conflict merging
- very stable
- very good Toolset
- strictly file based approach

SCM Comparison: Subversion (SVN)

- SVN is the unofficial successor of CVS and got created in 2000.
- Unique revision id over the whole repository
- commits are 'true atomic' functions
- versioned File-Metadata. File deletes und moves get historised as well.
- Subversion got some distributed features in 1.4 Though not for the history. Still no 'offline commit'

SCM Comparison: Visual SourceSafe

- Visual SourceSafe Microsofts 'old' SCM.
- purely file-based, access via a shared folder
- sporadically looses files if repo grews beyond 2GB
- branching is a real nightmare
- good integration into VisualStudio
- can be used standalone
- only runs under MS Windows
- costly

SCM Comparison: Team Foundation Server

- Microsofts successor of SourceSafe
- created by former Rational people
- only runs on MS Windows
- needs a license
- not very widely used

SCM Comparison: Mercurial

- better known as 'hg'
- Distributed SCM
- relatively slow protocol
- works well and is very stable
- written in Python

SCM Comparison: BitKeeper

- DSCM with strict central organisation
- formerly used for maintaining the Linux Kerne
- commercial product
- free for non-commercial use
- it's not allowed to reverse engineer it

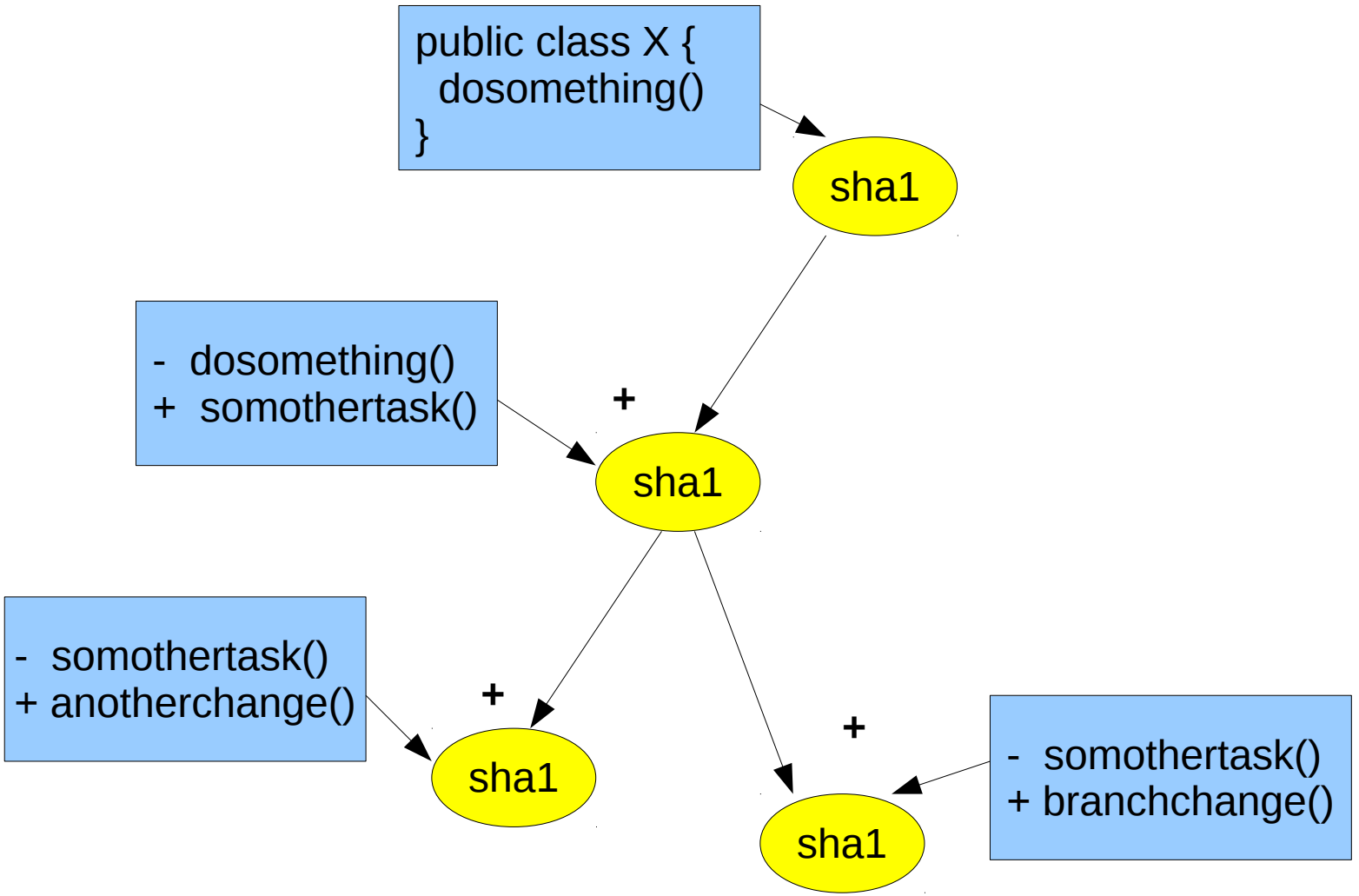
SCM Comparison: GIT

- GIT works similar to patch based systems
- GIT is de-centralised: all changes are also available 'offline'
- All 'patches' are available locally
- Commits are cryptographically strong

The basic GIT idea

Initial Checkin:

Some Change:



GIT Concepts: Object tree

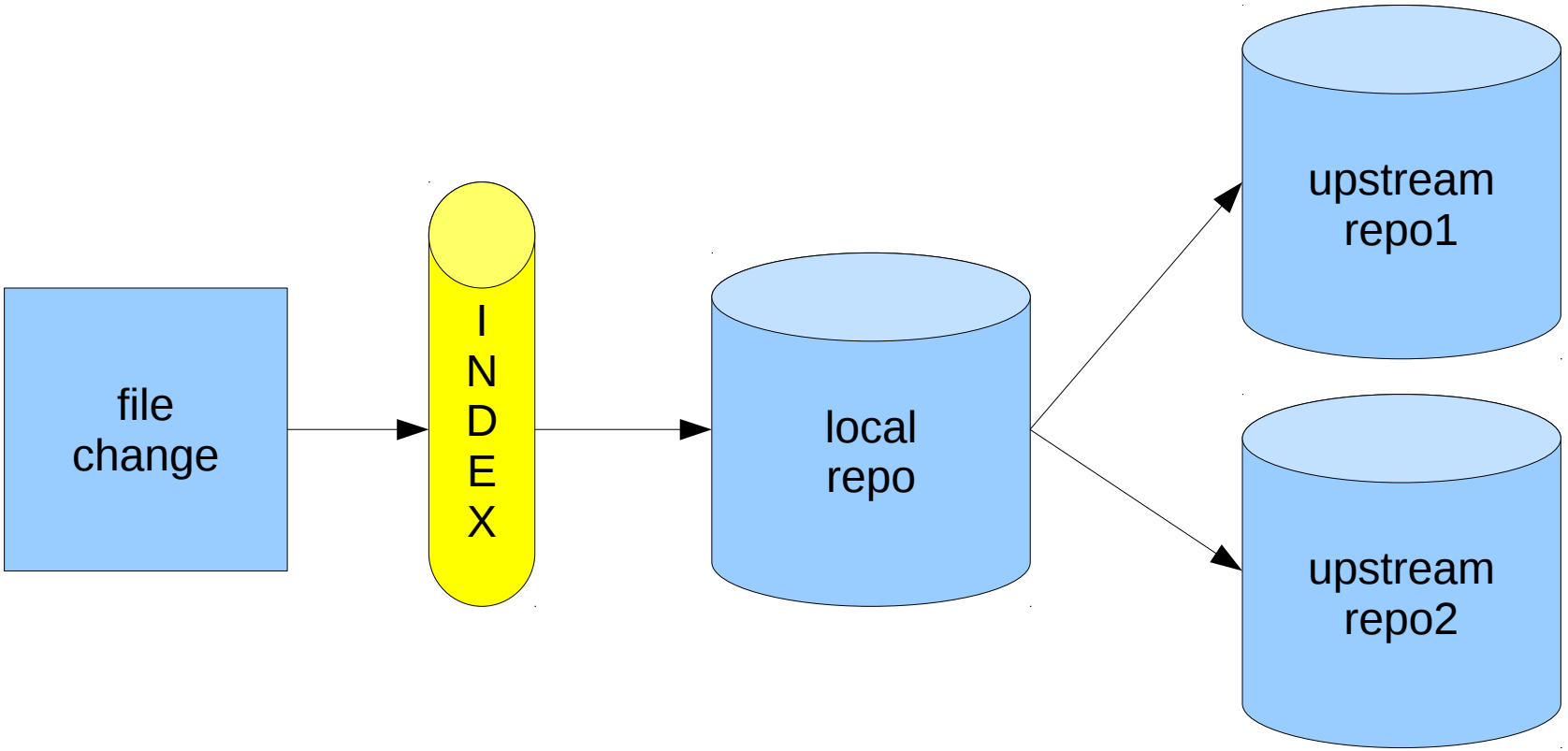
- GIT always tracks the whole repository
- GIT tracks a tree containing diffs with their 'parents' and commit Information
- .git/objects contains all those commits
- each commit has a unique sha1 containing the diff-object plus tree information, and further
- each commit has a unique sha1 containing the tree-object + commit info
- git 'packs' objects space optimized.

GIT Concepts: Commit Information

- Distinguishing between Author and Committer
- commit text
- author and commit date

GIT Concepts: the 'Index'

- GIT doesn't work directly against the Repository but has a 'preparation area' called 'Index'
- all changes prepared in that 'Index' will only get stored to the Repositories tree-objects with the 'commit'

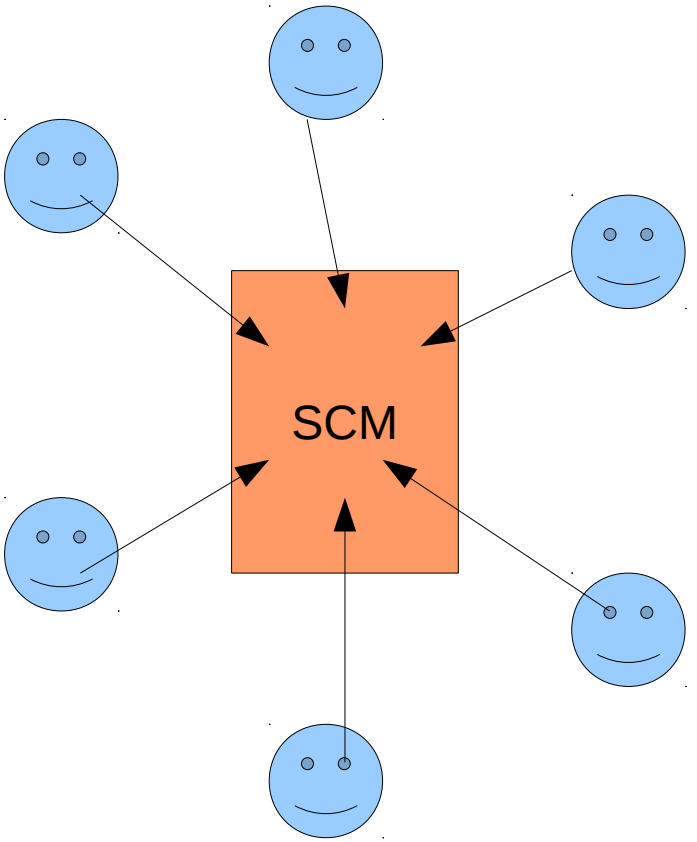


Some GIT history

- Junio C. Hamano is the maintainer of GIT since almost the beginning.
- GIT is 'Distributed', a new generation of SCMs. It requires a different mind-set in comparison to Client-Server based systems.
- Linus Torvalds developed GIT im April 2005, after lots of discussions regarding BitKeeper to maintain the Linux kernel

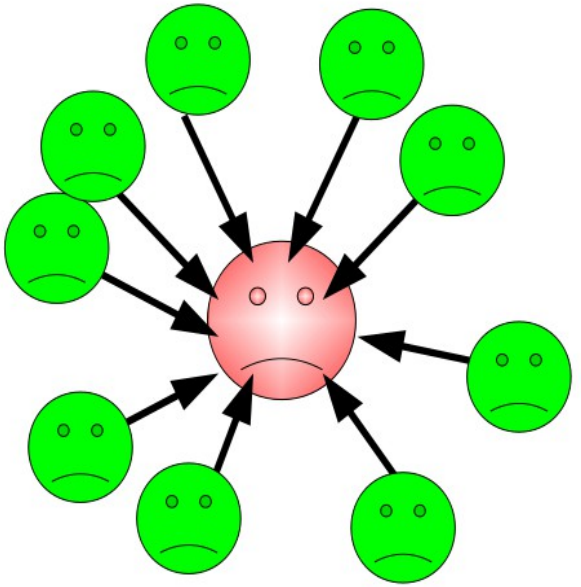
Topology of classical SCMs

- Client-Server Architecture
- All use the same Repository
- Each committer has write access
- Each committer checks in his changes himself – there is no review before checkin.



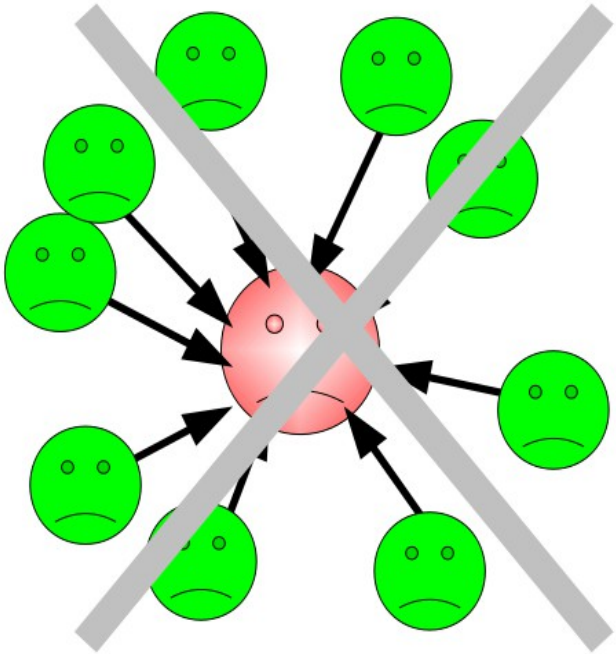
Linux kernel maintenance - old style

- Linus as single "Committer"
- All people sending patches to Linus
- Linus merged all Patches
- Linus maintained all changes himself



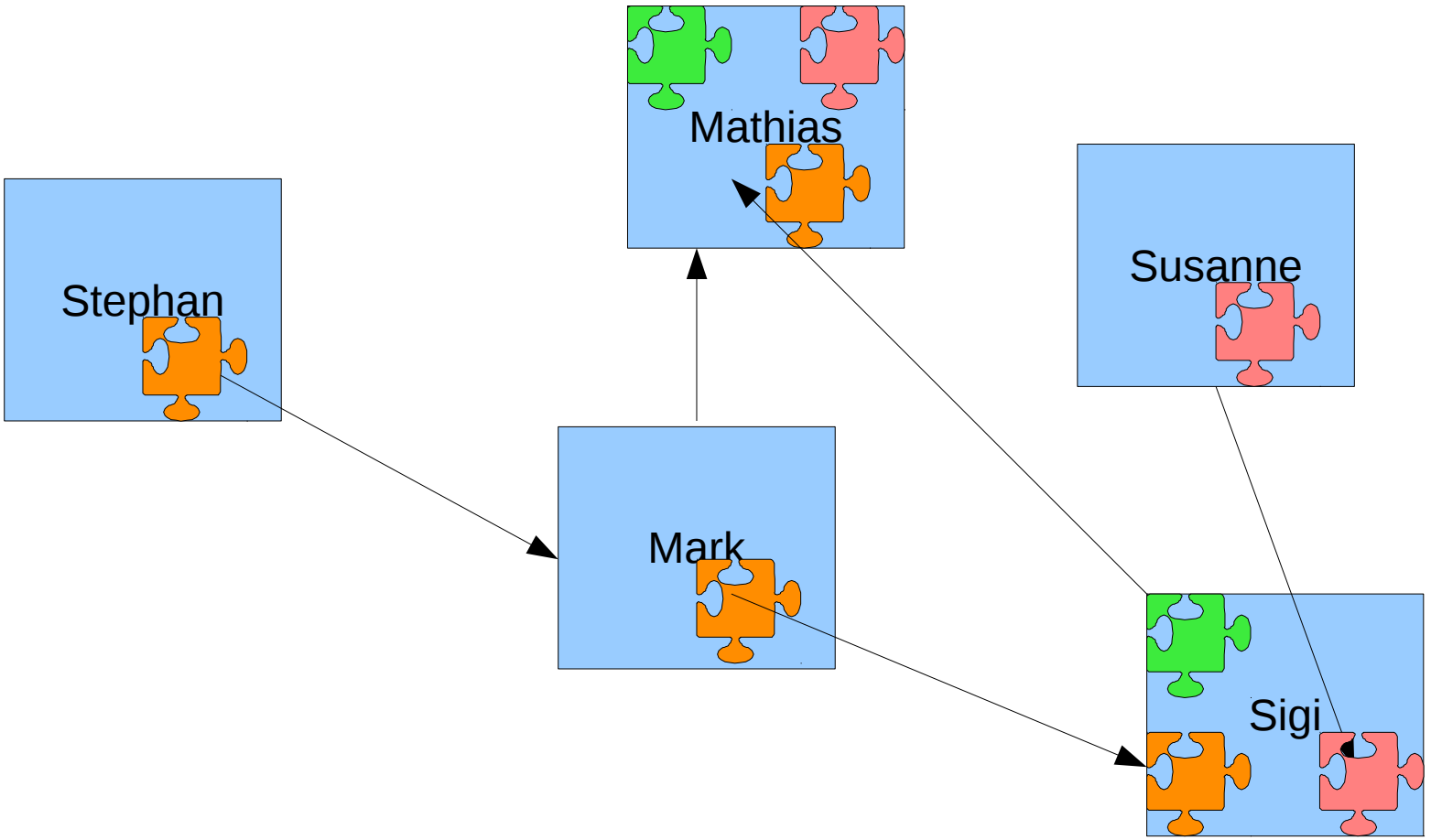
Problems of classical SCMs

- Classical Client-Server SCMs require a strict hierarchical organisation.
- Only works if the server is up and running
- Working offline is impossible
- In bigger projects you often face problems with access control



git – a stupid content tracker
Copyright © 2006 Junio C Hamano, All rights reserved.

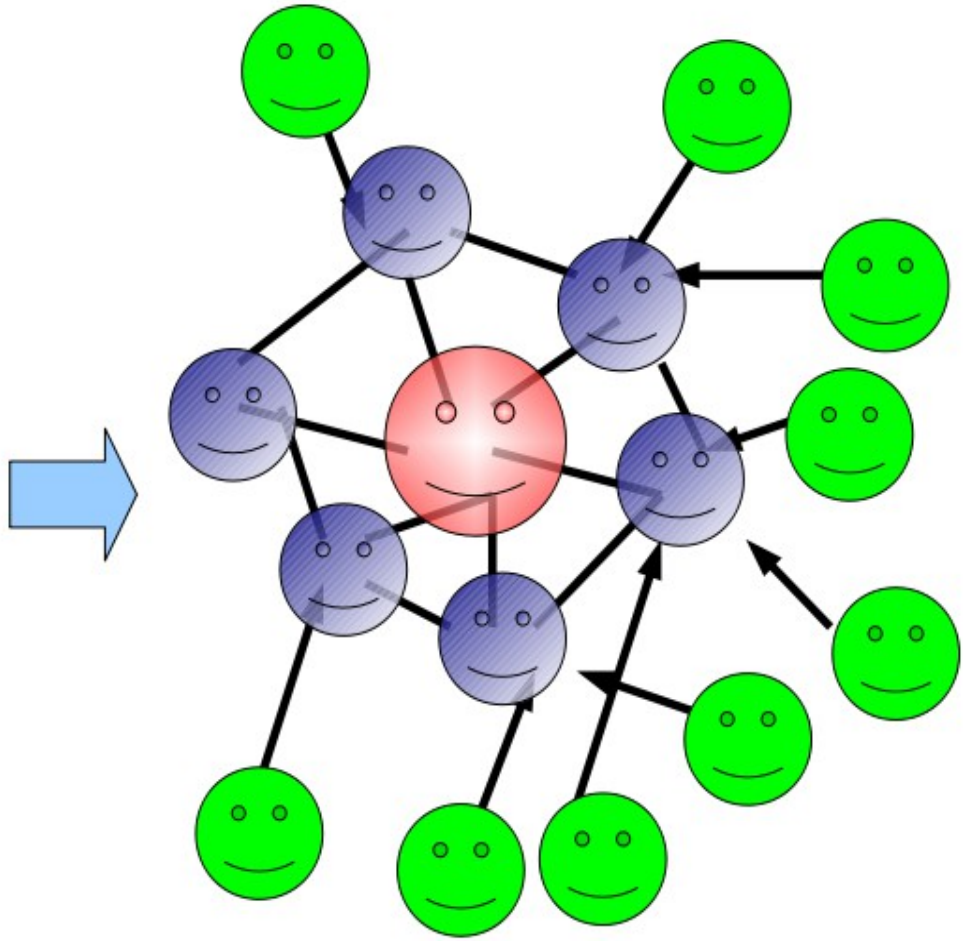
Topology of DSCMs



Maintaining Linux with 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

- Each user only writes to his **own Repository**
- Which means **no authorisation problem**
- **Good** changes get pulled actively
- **Author** of a change remains in the commit
- There is no technical master system
- Self organisation based on **Quality**
- Changes get reviewed a few times

Git Repository

- Pure **content-tracking** (NO file-tracking)
- list of changes (diffs)
- GIT collects Changes within a tree
- Direct Acyclic dependent Graph (**DAG**) of changes
- Each change has a **unique SHA1** hash
- That implies **Security**
- Additional **GPG** signature possible for tags
- **submodules** support
- all info under **.git** - no garbage in your work tree
- supports **various** transmission protocols

Weak spots of GIT

- optimized for whole **tree**s and not for single files
- **Annotation** is pretty **expensive**
- GIT manages **no authorisation**
- GIT manages **no file metadate**

transmission protocols

- **file** – access for local Repositories
- **ssh** – Authentication via Secure Socket Layer
- **http** – public internet. Write per webdav
- **git** – native protocol. extremely fast.
- support for **mixed operation mode** possible. E.g. read per http, write per ssh.

Vergleich SVN - GIT

- There is a comparison I wrote for the Apache Software Foundation: http://wiki.apache.org/couchdb/SVNvsGIT
- Both systems have strong and weak points!

GIT Commands common behaviour

- **git-command (obsolete style!)**
- or
- **git command**
- GIT commands most times work against the local 'Index'
- **HEAD** always points to the last revision of your work
- **man git-command** shows the help

configure GIT

- **git config** allows to read and set various configuration
  - `git config --global user.name „My Name“`
  - `git config --global user.email „name@domain“`
- Only for a single repo:
  - `git config user.email „name@domain“`
- Local config in ./.git/config
- Global config in ~/.gitconfig
- vi .gitignore

Working with local GIT Repos

- **git-init**
- **git-add**, **git-rm** to stage your local changes in the 'Index'
- **git-reset** to reset the 'Index'
- **git-commit** to propagate your staged changes from the Index to the local repository
- optional **git-push** to a 'public' Repository

Cloning an existing Repo

- **git-clone**
- **git-status**
- **git-add**, **git-rm** of local changes to the Index
- **git-commit** to transfer the changes from the Index to the repository
- **git-push** for transferring the changes to the upstream 'origin' repository

Working with Branches

- GIT makes **branching** a breeze
- and even more the **mergen**
- **maintenance**-branch
- **feature**-branch

Git branch Befehle

- **git-branch** [branchname] creates a new branch
- **git-checkout** [branchname] switches the workspace to the given branch
- **git-rebase** allows 'fast forward' of patches
- **git-tag** creates a unique 'name' for a SHA1 commit
- **git-checkout** [tagname] gets a tag from the repo
- **git-cherry-pick** [SHA1] applies a single commit to the current HEAD

Mergen mit Git

- GIT uses a **3-way merge**
- **git-merge** [branch1] [branch2]
- **git-pull** [remotebranch]
- **git-mergetool** helps to resolve conflicts
- Conflicts need to get resolved manually with git-add / git-commit
- Benefit of the internal **diff** management against classical SCMs: easier merging of branches
- **git-bisect** for searching errors

More important GIT commands

- git-diff - shows changes between versions
- git-log - shows the commit history
- git-status - shows current changes
- git-push - transfers changes to other Repos
- git-reset - resets the 'Index'
- git-revert - creats and commits an 'undo patch'
- git-ls-files - lists files in Index and Tree
- git-ls-tree - shows the content of a Tree Objektes
- git-repack - packages multiple diffs to save space
- git-gc - deletes unused Blobs (garbage collect)

Graphic Tools

- **gitk** - is a perl/tcl gui for browsing the repository history
- **git-gui** - allows graphical git-status, git-add, git-commit, etc
- **egit** – pure Java based Implemenation e.g. used in Eclipse and IDEA

git-rebase

- git rebase applies own changes on the top of merged in changes
- git rebase otherbranch
  - detect commonly used parent
  - store away the diff up to the common parent and rollback those own changes in the tree
  - merge the otherbranch into the current tree
  - re-apply the stored away changes 'on-top' of the merged changes.
- ATTENTION: history rewrite!

GIT und Line Endings

- GIT originally only treated 'native' Line Endings
- a configuration got only added pretty late
- `git config --global core.autocrlf input|native|true|false`
- http://help.github.com/line-endings/
- pre-commit hook with automatic dos2unix based on file extension

git stash

- **git stash** or
  git stash save saves away local (uncommitted) changes (working directory and Index) into a stack
- **git stash pop** gets back those stored changes

My private Git

- Even if the company or the project maintains another canonical repository, it can still be useful to locally work with git
- Beneficial for **feature branch**es.
- Beneficial for small **collaboration groups**

Git „on top“

- create a local GIT repository without storing this info in SVN
- **git-init** in your SVN checkout folder
- adopt **.gitignore** and **.svnignore**
- **git-branch** feature1
- **git-checkout** feature1
- **git-add**, **git-commit**
- intermediate bug fixing -> **git-checkout master**
- after that **git-checkout feature1**
- finally **checkin in** SVN
- update of the master branch in your local GIT repo

SCM Connectivity

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

Working with git-svn

- **git svn clone URL** gets changes from a SVN repo and creates a local GIT repo
- normal workflow with git-add, git-commit etc
- **git svn dcommit** commits you GIT changes to the SVN server
- **git svn fetch** gets updates from SVN
- ATTENTION: be careful when switching branches
- ATTENTION: will most times lead to history rewriting!
- Thus it's not possible to share this repo easily with others

GIT as canonical SCM

- 'SVN like' central approach
- different people pushing to one single cannonical GIT repo
- https://cwiki.apache.org/confluence/display/DeltaSpike/Suggested+Git+Workflows
- history rewriting must be disabled on the server!
- **git pull --rebase** to prevent 'auto merges'
- do temporary collaborations in an own GIT repo
- always review the history **before** pushing your work upstream

GIT workflow for canonical repos

- ALWAYS work in your own branch (e.g. 'strub')
- git checkout master
- git pull --rebase
- git checkout strub
- git rebase master
- git checkout master
- git merge strub
- git push origin master:master

Gitolite

- Gitolite allows to specify different privileges for **push** operations.
- There is no way to prevent reading the whole repo
- https://github.com/sitaramc/gitolite
- gitolite is implemented via hooks in the GIT lifecycle

Maven und GIT /1

- Apache Maven supports GIT since 2007
- ```
  <scm>
      <connection>scm:git:
  https://git-wip-us.apache.org/repos/asf/incubator-deltaspike.git
      </connection>
      <developerConnection>scm:git:
  https://git-wip-us.apache.org/repos/asf/incubator-deltaspike.git
      </developerConnection>
      <url>https://git-wip-us.apache.org/repos/asf/incubator-deltaspike.git
  </url>
  </scm>
  ```
  ```

Maven und GIT /2

- The maven-release-plugin also supports GIT:
- <plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-release-plugin</artifactId>
  <version>2.2.2</version>
  <configuration>
  <pushChanges>false</pushChanges>
  <localCheckout>true</localCheckout>
  </configuration>
  </plugin>

legal aspects

- Git is released under GPLv2
- egit is EPL
- maven-scm-providers-git under ALv2

Links

- http://git-scm.com
  Git wiki and official page
- http://youtube.com/watch?v=4XpnKHJAok8
  Linus on Git
- http://www.youtube.com/watch?v=8dhZ9BXQgc4
  Randal Schwartz about git
- http://help.github.com
  github tutorial and common GIT stuff
- http://www-cs-students.stanford.edu/~blynn/gitmagic/intl/de/
  good tutorial
- http://wiki.apache.org/couchdb/Git_At_Apache_Guide
  http://wiki.apache.org/couchdb/SVNvsGIT
- http://code.google.com/p/msysgit/
  GIT for Windows

Q&A