Push up your code – next generation version control with (E)Git

Dominik Schadow
Senior Consultant
Application Development

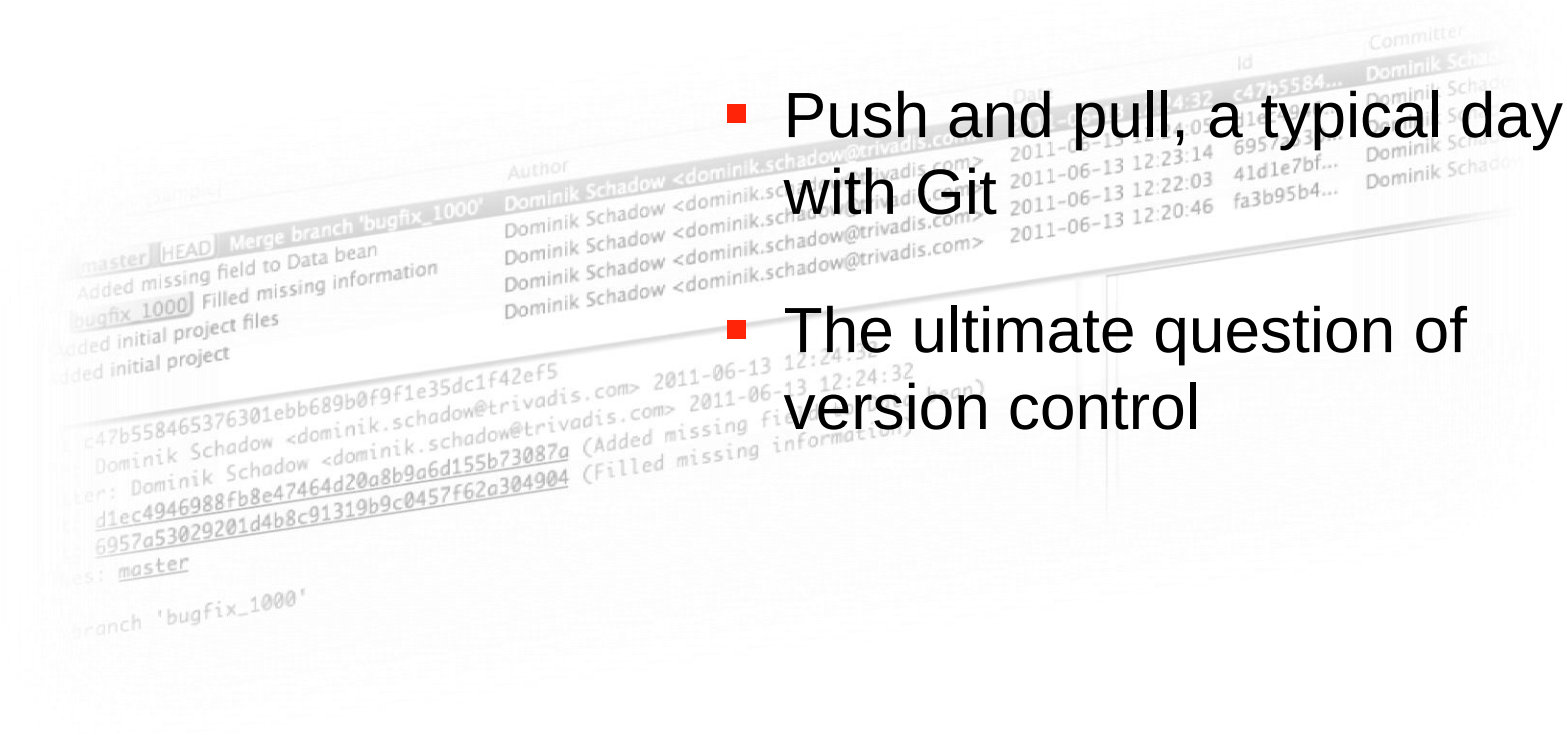
dominik.schadow@trivadis.com

Java Forum Stuttgart, 07.07.2011

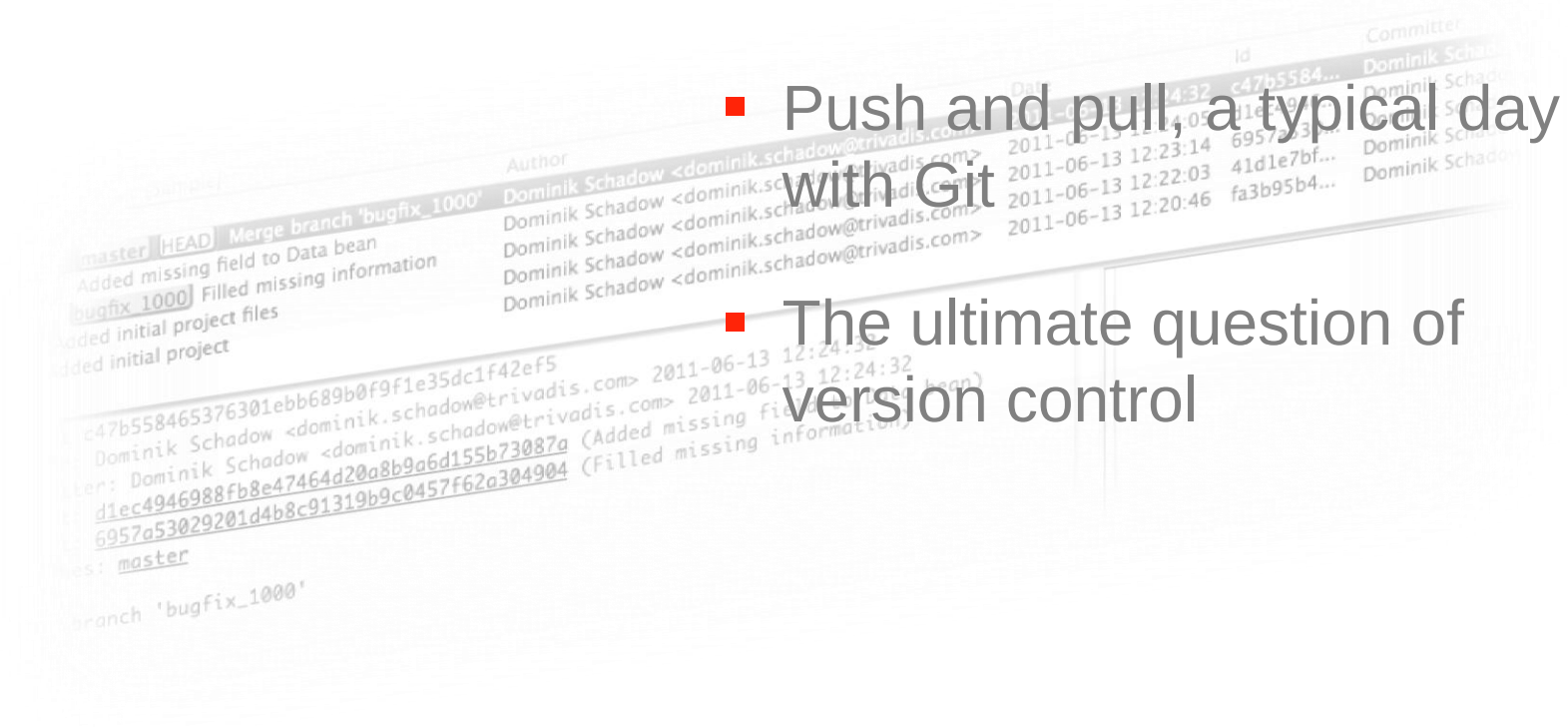
trivadis
makes IT easier.

Agenda

- Almost all about Git and EGit
- Push and pull, a typical day with Git
- The ultimate question of version control



Agenda



- Almost all about Git and EGit
- Push and pull, a typical day with Git
- The ultimate question of version control

Subversion and CVS have many disadvantages

- Creating a branch is easy and fast
  - Merging is a pain (most of the time)
  - All branches are shared, no local (private) branches
- Central repository server makes backups easy
  - Everybody is working against the same repository
  - Clients require server connection for most operations
- Performance is good for certain operations
  - Slow merge, diff or switch operations
  - Slows down as the project (history) grows larger

Git was created in the Linux community

2005	Git development starts in the Linux (kernel) community by Linus Torvalds
2006	JGit development starts, a 100% pure Java reimplementation of the Git version control system
2009	EGit/ JGit move to Eclipse, first projects migrate to Git
2010 (Sep)	EGit/ JGit 0.9.1
2011 (Feb)	EGit/ JGit 0.11.1
2011 (Jun)	EGit/ JGit 1.0 with Eclipse 3.7
2012 (Jun)	EGit/ JGit 2.2 with Eclipse 3.8

JGit and EGit are official Eclipse projects

- The original **Git**
  - Original version developed by the Linux community
  - Distributed under the GNU General Public License (GPL)
- **JGit** is a lightweight Java library implementing Git
  - JGit library can be found in many Java based products
  - Plug-ins for Eclipse and NetBeans IDE, Hudson CI server, Apache Maven, and Gerrit Code Review
  - Distributed under the Eclipse Distribution License (EDL)
- **EGit** is the Eclipse team provider and uses JGit
  - No team provider trouble as with Subversion
  - Normally no command line required
  - Distributed under the Eclipse Public License (EPL)

Git is a **D**istributed **V**ersion **C**ontrol **S**ystem (DVCS)

- Git clients fully mirror the repository
  - Every clone is a complete backup
    - Git always clones the entire repository
    - No partial checkout possible
  - The whole repository is available locally
    - Entire development history
    - Complete repository with all branches, not only the latest snapshot
- No network connection required
  - Most operations, except *push/pull* and *fetch*, work offline
    - Much better performance
  - No central server is required
    - Local repository for private development
    - Clients can directly communicate with each other

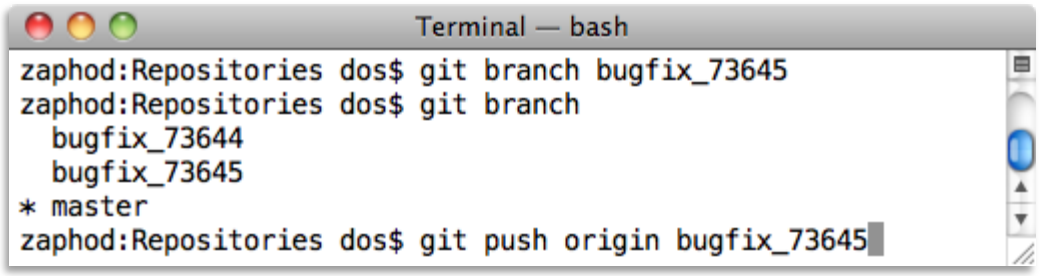
Branching and merging is easy and fast

> *„In Git it‘s common to create, work on, merge, and delete branches several times a day.“*
> http://progit.org/book

- Branching and merging are an essential Git concept
  - Create local branch for each feature/ bug fix you work on
  - You can have many feature branches at any time
    - Easy to switch between them
    - No mix up of changes in the same branch
  - History-aware merging capability
    - Auditing of branch and merge events

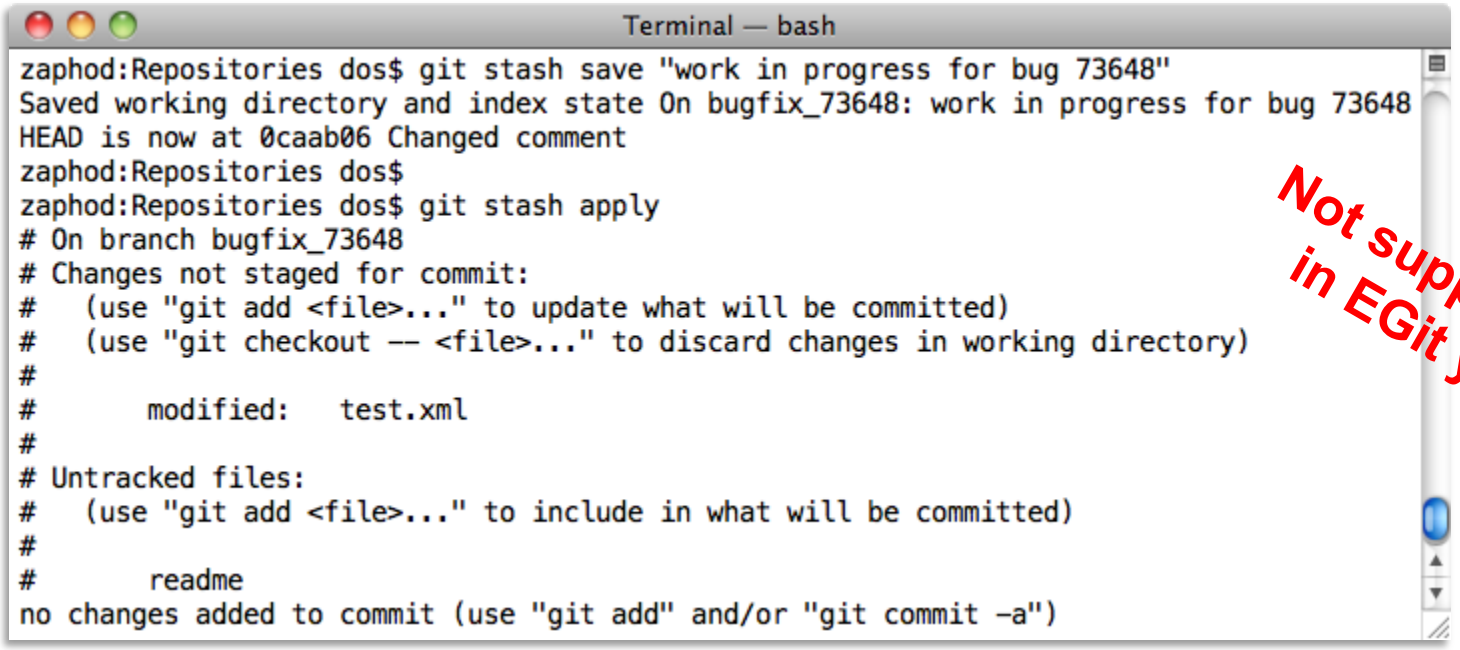
The default ‘trunk’ is called ‘master’ in Git

- All branches are local after creation
  - Extremely fast, no network communication required
  - Every developer’s working copy is a private branch
- Easy to share a branch (or tags) with other developers
  - But most branches live only for a short time locally
  - Push to share
    - **git push (remote) (branch)**



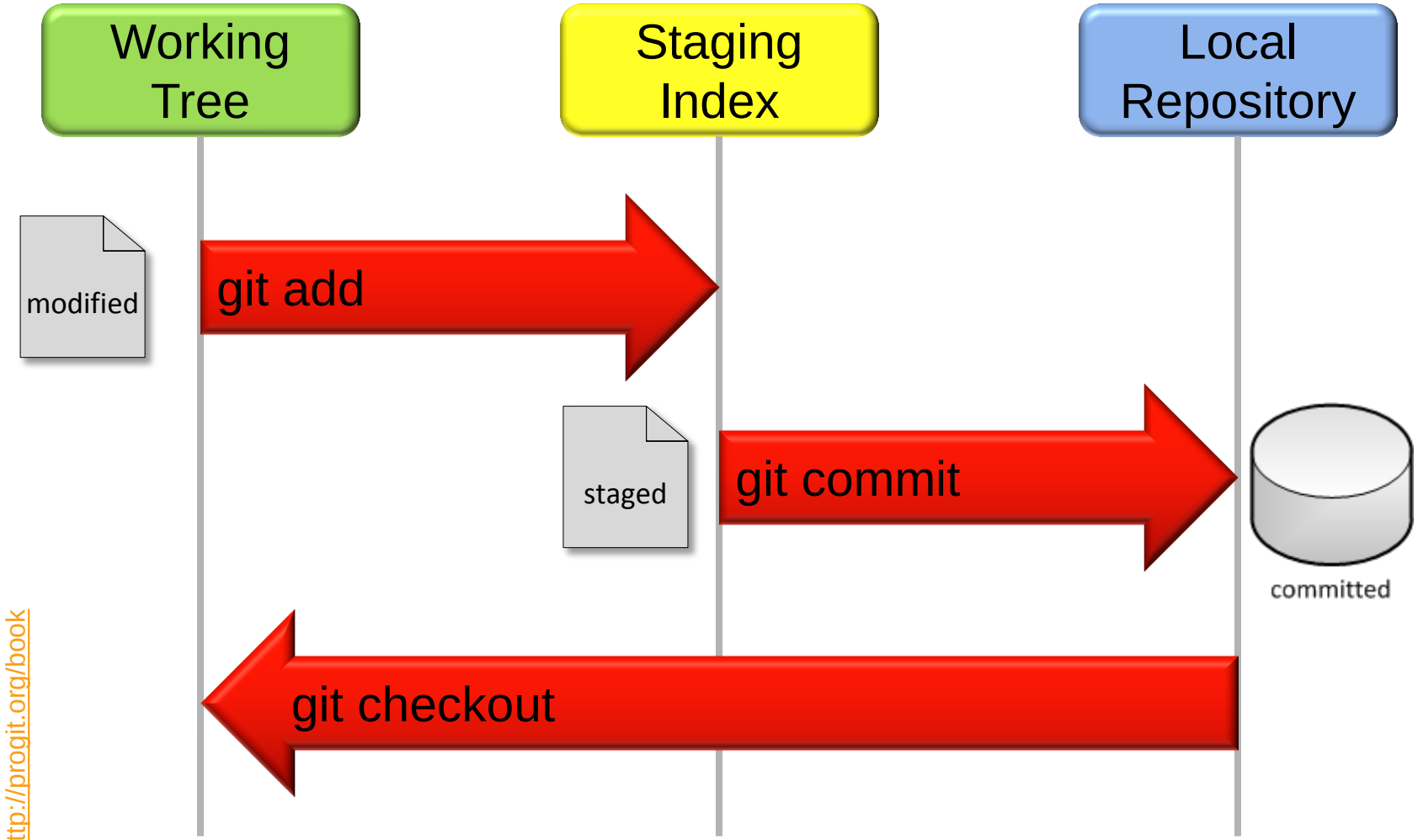
Store your working directory and revert

- Use **git stash** to record current working directory state
  - Saves current state of work
  - Resets working tree/ index to match latest version of current branch (a clean workspace)
  - Re-apply it at later to continue your work



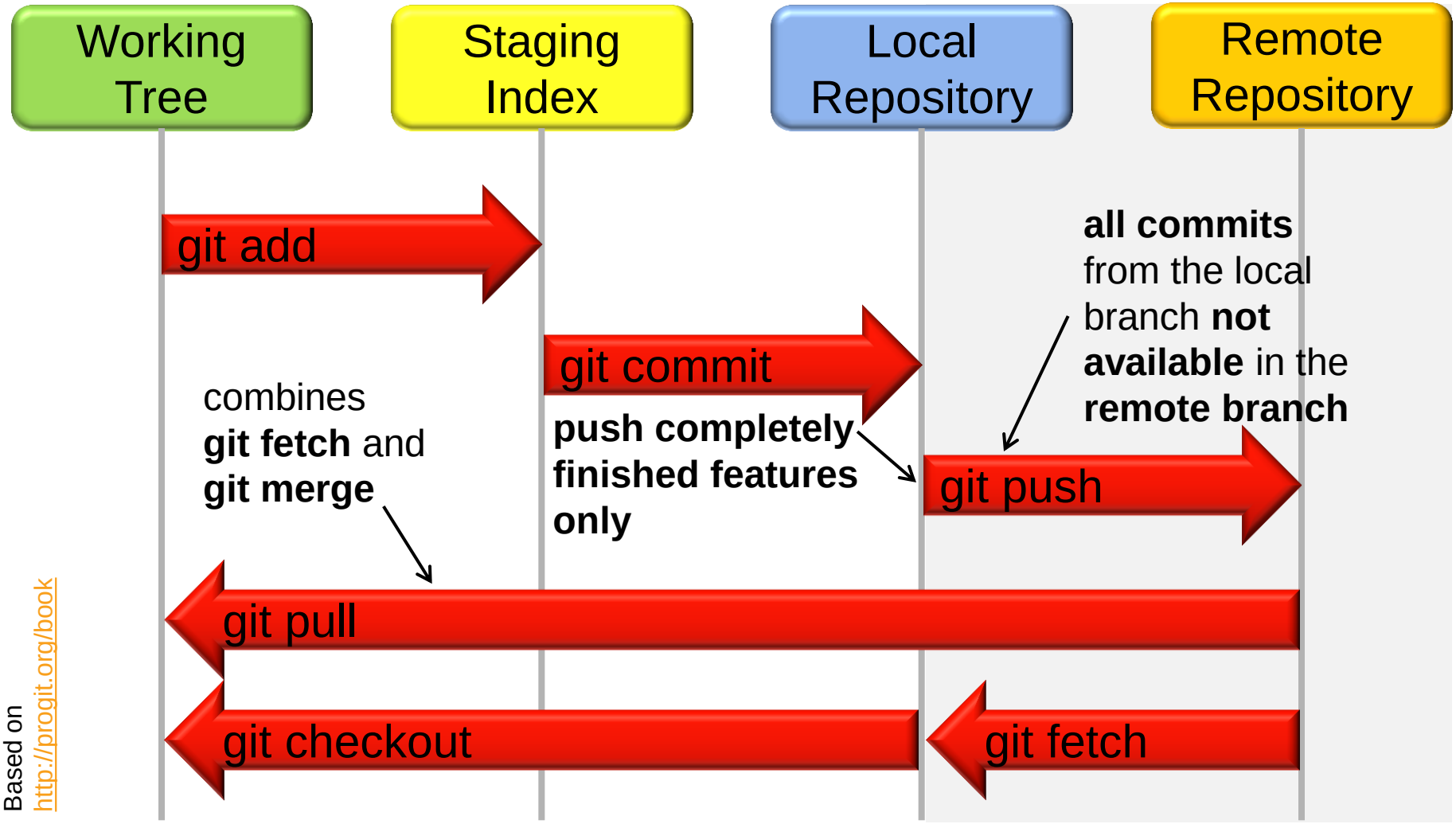
Not supported in EGit yet

There are three main states/ sections in a Git project



Based on http://progit.org/book

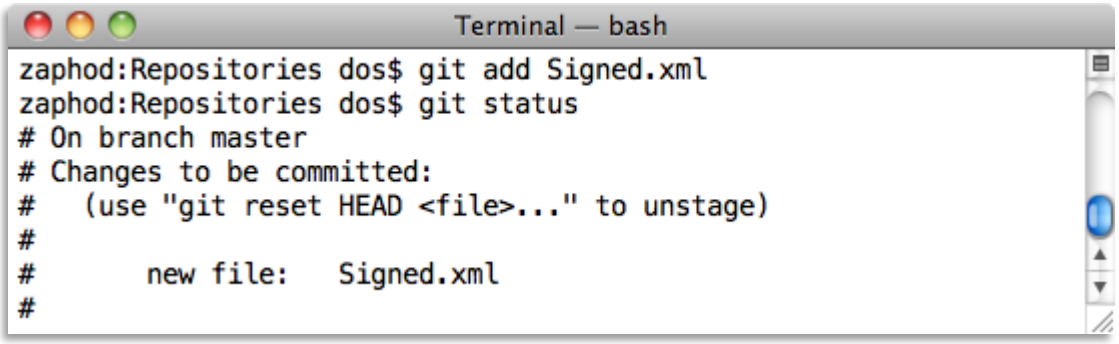
Changes flow between repositories by push and pull



Based on http://progit.org/book

The index is a staging area for the next commit

- Index is changed via **git add**

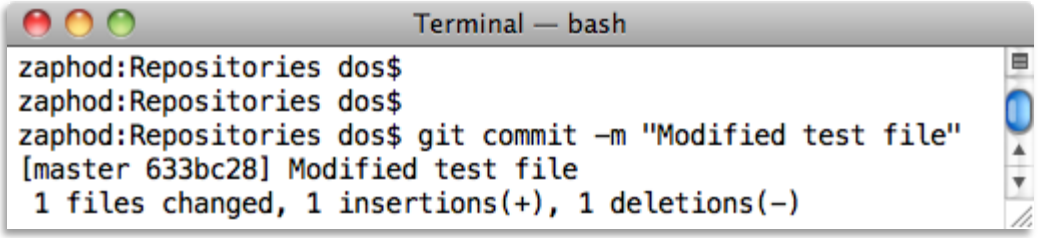


- State of the index becomes the tree of the next commit
  - Index provides an extra layer of control
  - Index is like an active changeset

Git tracks objects by their hash value

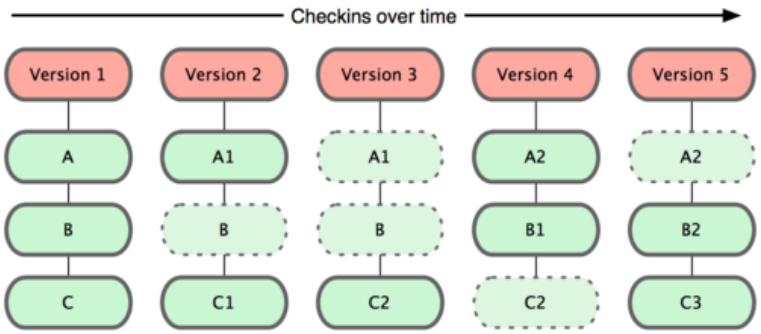
- Each blob is identified/ named by a SHA-1 hash
  - Git automatically computes the hash
    - Hash input is the objects content
    - Tamper-proof signature as a bonus
  - Blob does not contain any metadata

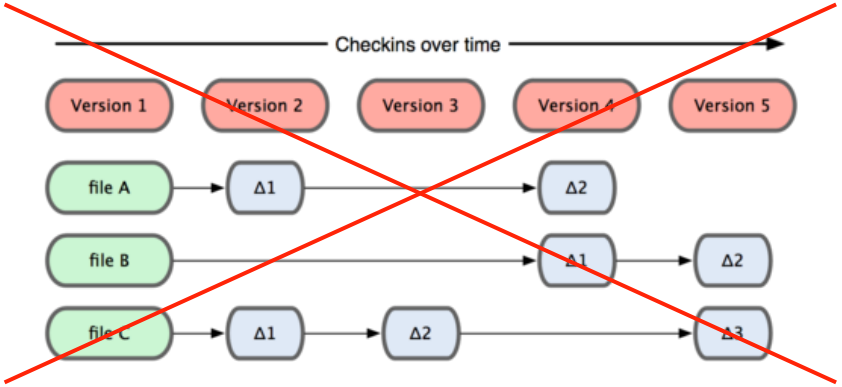
- Path and filename information is not considered
  - A renamed file is still linked with the original version
  - Sometimes problems with binary files
    - Even a small change might create a whole different hash
    - Relationship between new and original file might be lost



The append-only object database

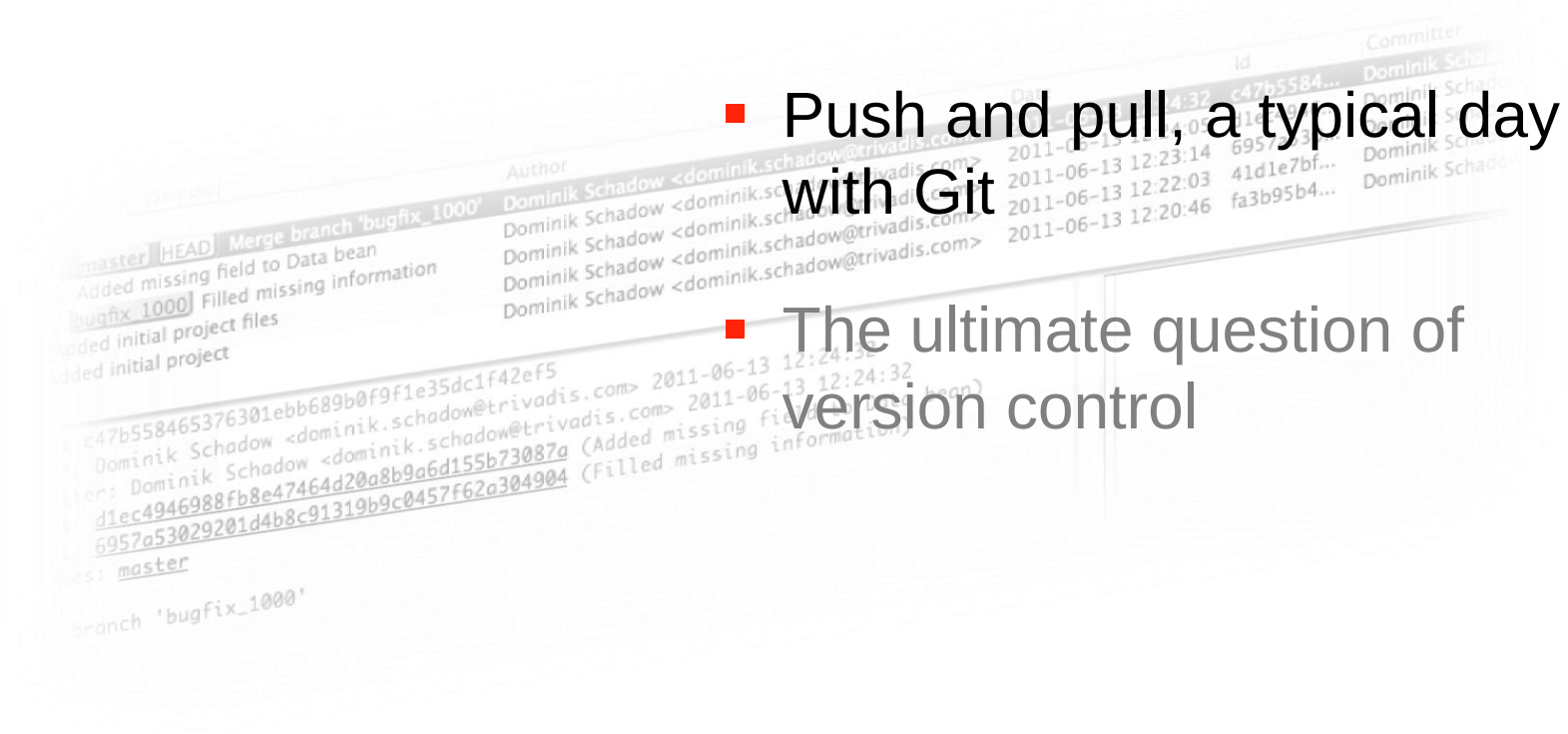
- Git stores each revision of a file as a unique blob object
  - Relationships between the blobs
    - Can be found through examining the tree and commit objects
  - Newly added objects are stored in their entirety
    - Git saves states, not deltas as Subversion
    - Uses zlib compression





Based on http://progit.org/book

Agenda

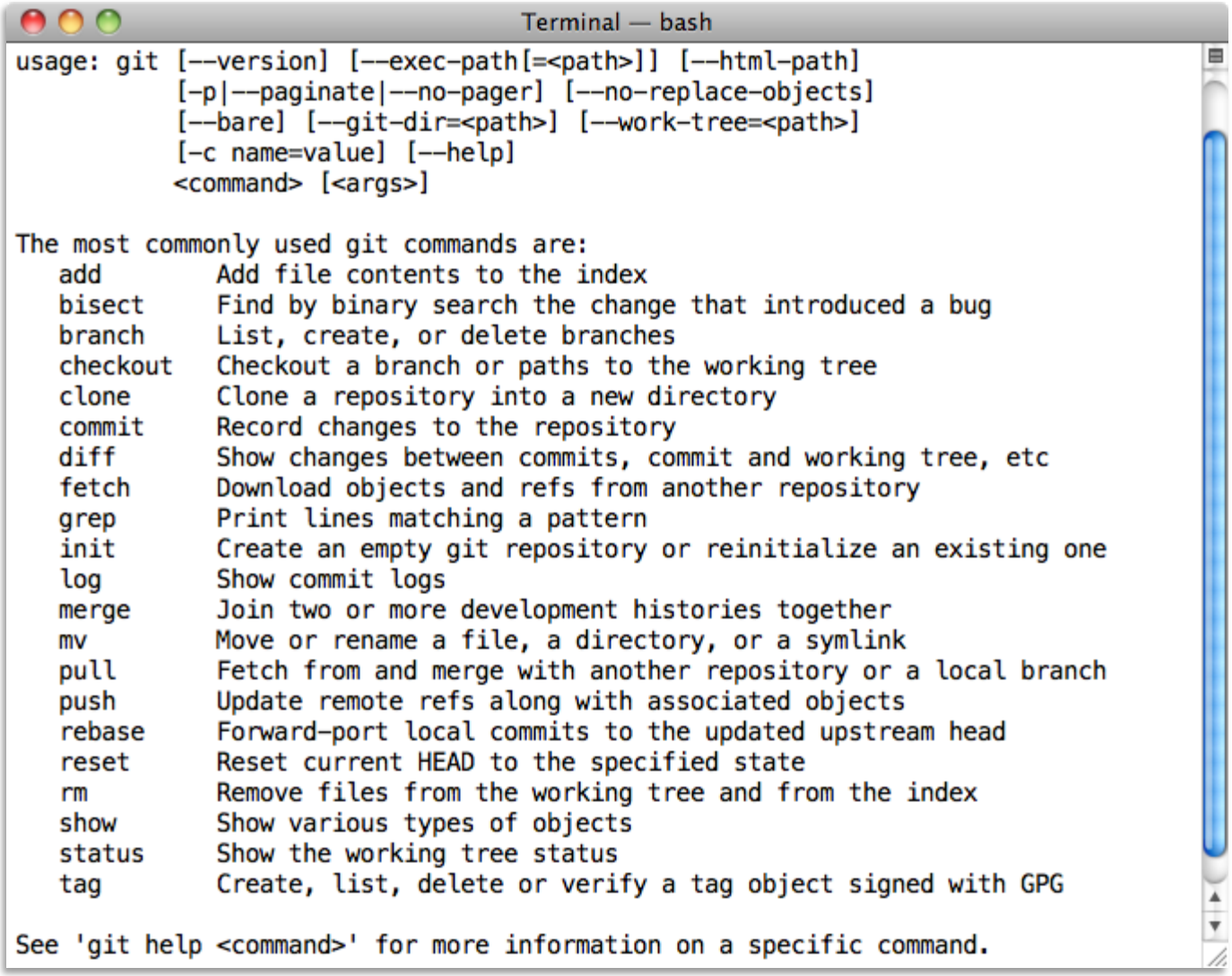


- Almost all about Git and EGit
- Push and pull, a typical day with Git
- The ultimate question of version control

Git command line

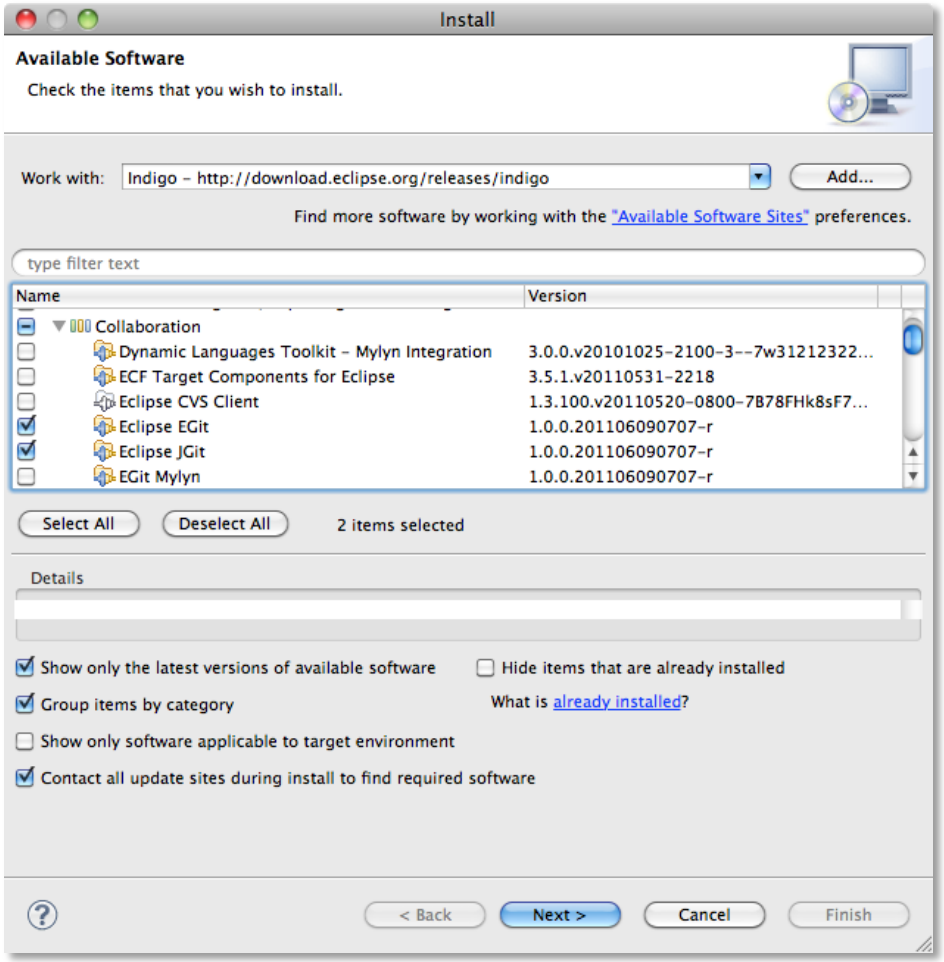
- Git is available for Linux, Mac OS X and Windows
  - Windows command line is a little bit slower
- Clients/ command lines are in different development stages
  - Generally better and tighter integration on Linux and Mac OS X
- Some initial configuration required
  - Creates the *.gitconfig* file in your home directory
  - Via command line or Eclipse preferences

Git commands



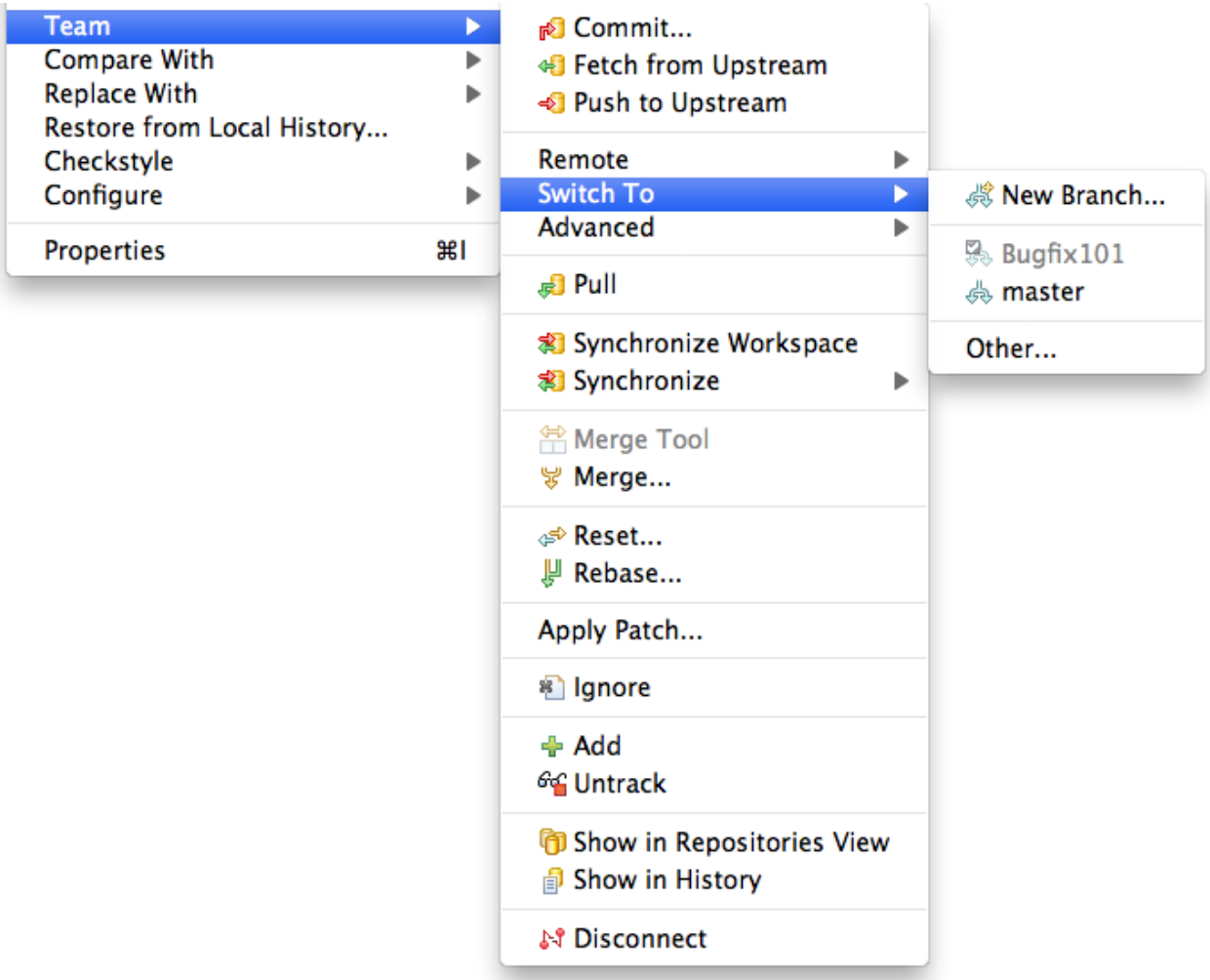
EGit/ JGit installation via Eclipse update site

- Git command line is not required
  - But Plug-ins do not provide command line interface
- Install via update site
  - Eclipse EGit
  - Eclipse JGit



Before Indigo http://download.eclipse.org/egit/updates

EGit provides almost everything you need

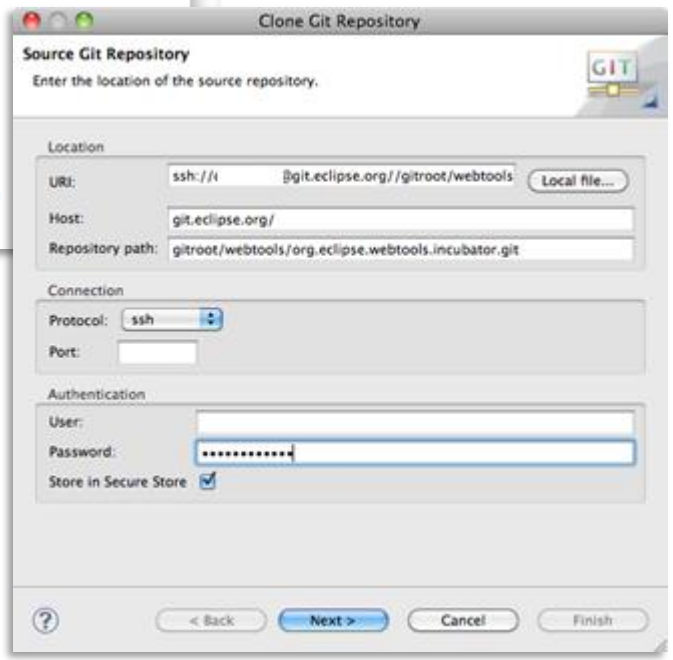


Cloning an existing repository

- Git clone automatically names the clone **master**
  - **master** is based on the remote **origin** branch
  - Creates a new directory
    - Using the Git repository name as directory name
    - Use optional *directory* parameter to specify a different name
- All its data is pulled to the local repository
  - A pointer to its master is created
  - Never modify the single **(one and only one)** .git directory
    - That is the Git repository
    - Exists only once in your repository root
    - Files/ directories under the parent of .git are the working tree

Initialize a new repository or clone an existing one

```
Terminal — bash
zaphod:eclipse dos$ git clone ssh://        @git.eclipse.org/gitroot/webtools/
org.eclipse.webtools.incubator.git source
Cloning into source...
Password:
remote: Counting objects: 38227, done.
remote: Compressing objects: 100% (12559/12559), done.
remote: Total 38227 (delta 18699), reused 31072 (delta 15256)
Receiving objects: 100% (38227/38227), 15.96 MiB | 377 KiB/s, done.
Resolving deltas: 100% (18699/18699), done.
```



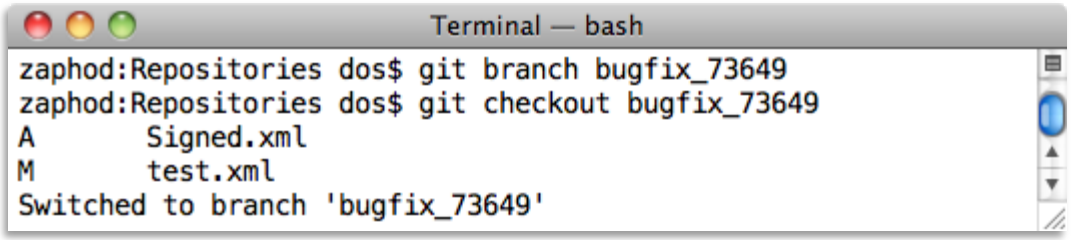
Git supports many different protocols:
file, ftp, **git**, http, https, sftp, ssh

faster, more efficient, but read-only

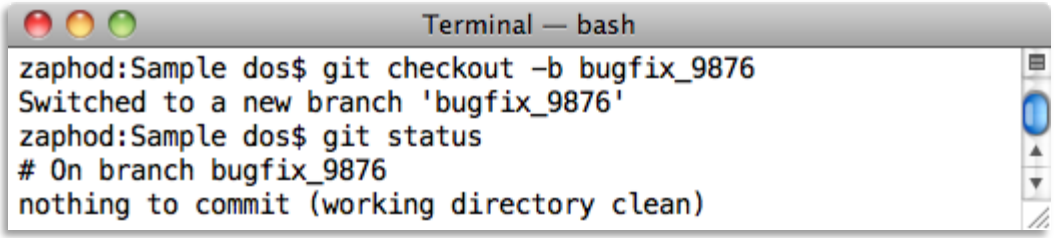
```
Terminal — bash
zaphod:~ dos$ cd Repositories/
zaphod:Repositories dos$ git init
Initialized empty Git repository in /Users/dos/Repositories/.git/
zaphod:Repositories dos$
zaphod:Repositories dos$
```

Creating new branches

- Creating a new branch creates a new pointer (fast!)
  - Points to the same commit currently working on
  - Switch to the new branch with checkout

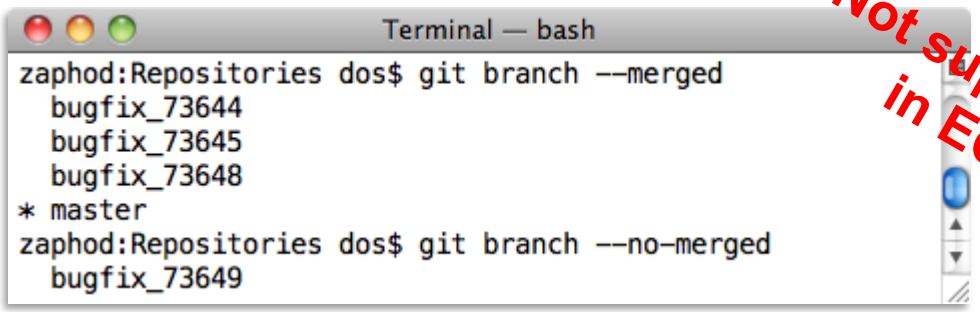


- Or create and switch with a single command



Merging is trivial in Git

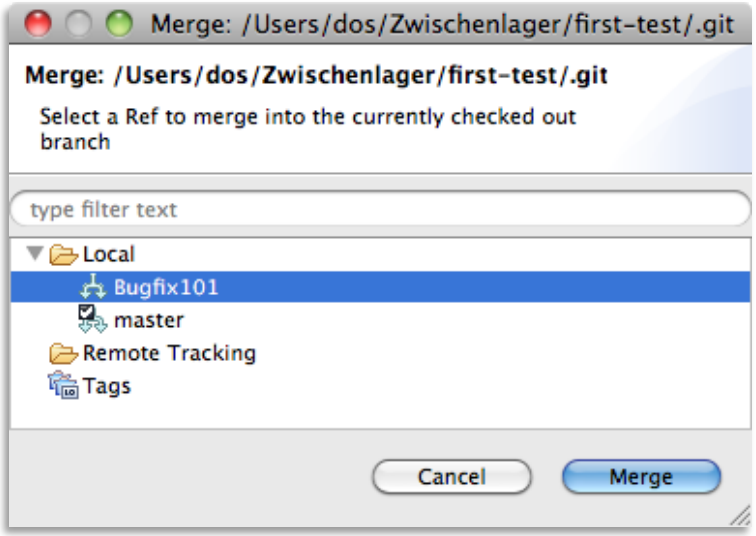
- Each changeset tree node
  - Contains a pointer to its previous node
    - Back to the first commit
- Git knows what changes need to be made
  - And at what point in history they need to be applied
  - Automatically merges the given branch into the active one
- Listing the merged and unmerged branches



Not supported in EGit

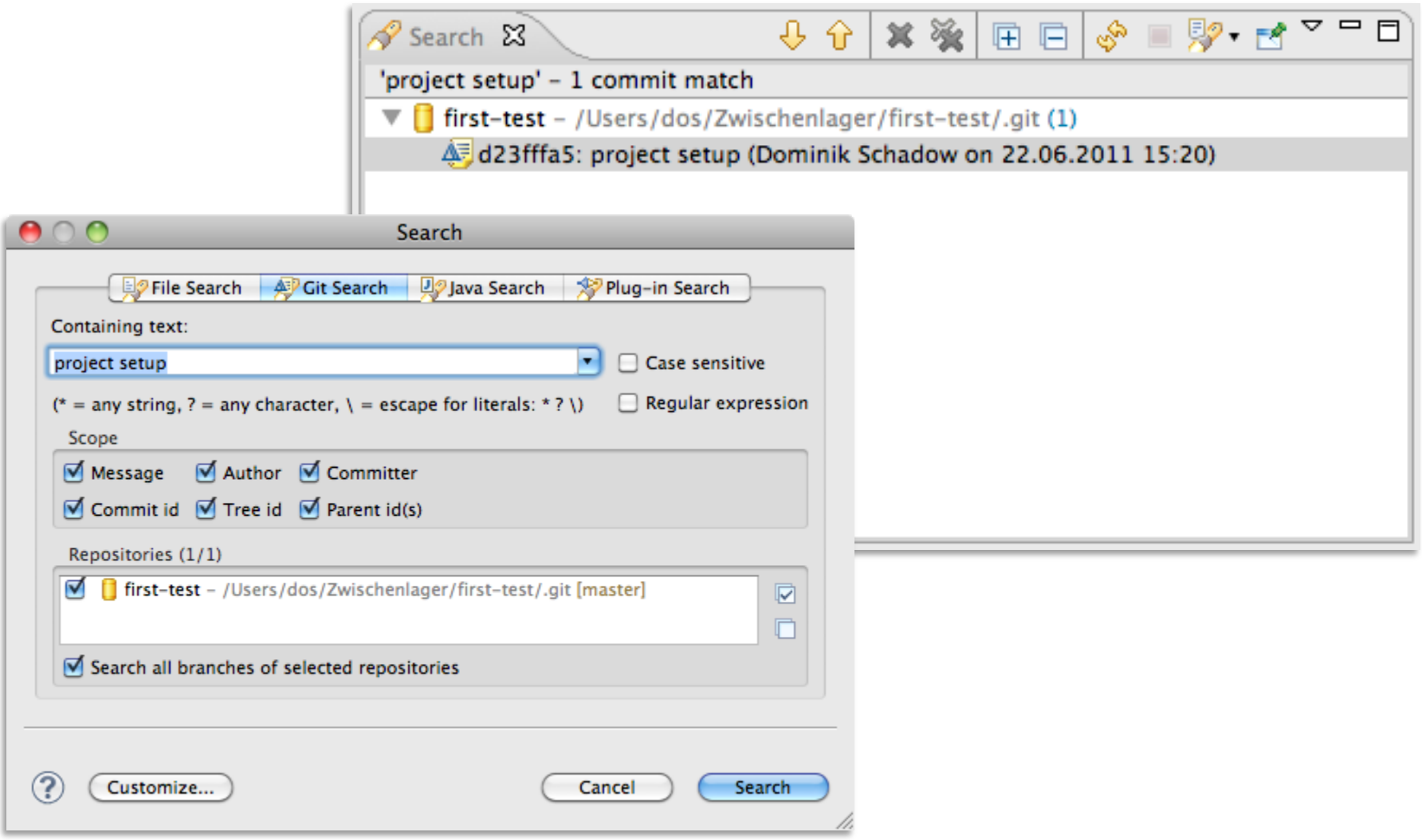
Switch to the branch to merge the changes in

- Use **git merge** and select the branch to integrate
  - **Fast-forward merge**
    - Only the other branch changed
    - No merge operation required
  - **Three-way merge**
    - Both branches changed
    - Don’t expect miracles, conflicts happen: Resolve with merge tool or manually

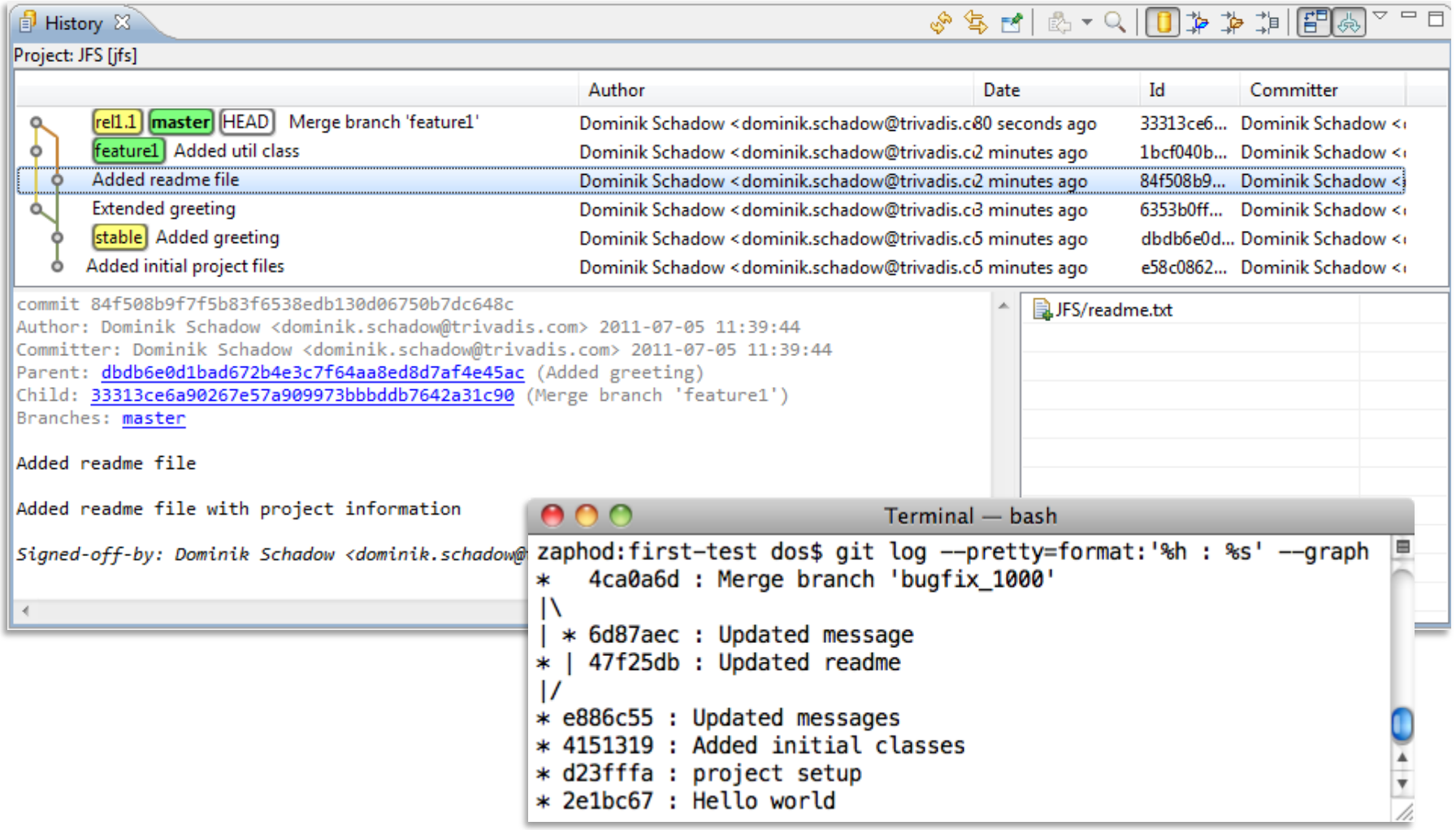



```
Terminal — bash
zaphod:Sample dos$ git checkout -b bugfix_1000
Switched to a new branch 'bugfix_1000'
zaphod:Sample dos$ git checkout master
Switched to branch 'master'
zaphod:Sample dos$ git merge bugfix_1000
Merge made by recursive.
 Files/Readme                    |    4 +++-
 src/com/trivadis/jfs/Main.java |    2 +-
 2 files changed, 4 insertions(+), 2 deletions(-)
```

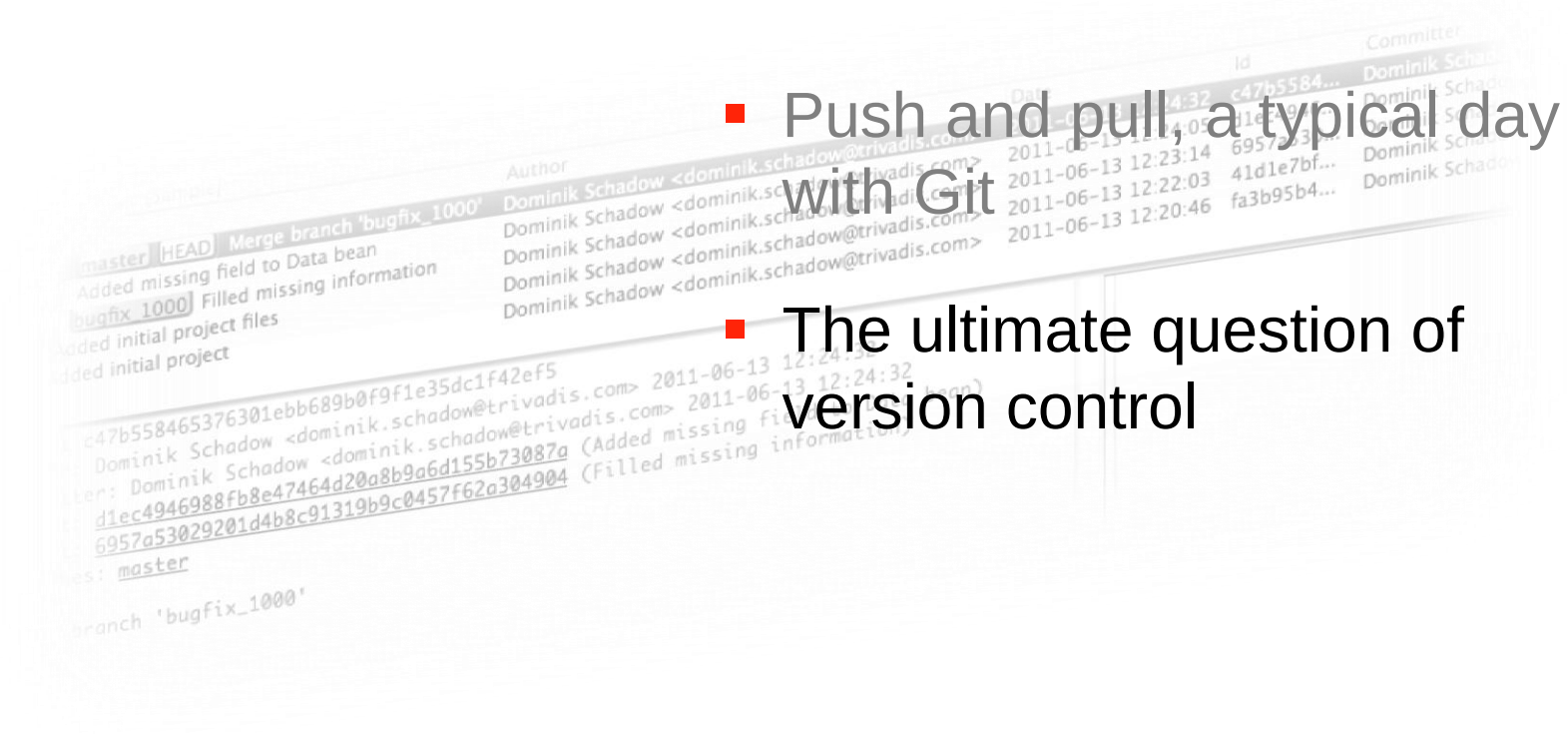
Searching for Git commits



EGit History view and the Git log

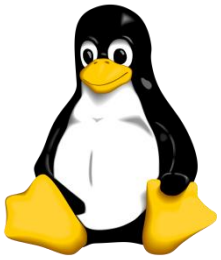


Agenda



- Almost all about Git and EGit
- Push and pull, a typical day with Git
- The ultimate question of version control

Git command line interfaces and tools



- gitg http://trac.novowork.com/gitg
- giggle http://live.gnome.org/giggle

- Git for OS X http://code.google.com/p/git-osx-installer
- GitX http://gitx.frim.nl

- cygwin http://www.cygwin.com
- msysGit http://code.google.com/p/msysgit
- TortoiseGit http://code.google.com/p/tortoisegit

Git IDE integration

- Useable version since EGit 0.11
- Stable version available with Eclipse Indigo
- Updates 1.1/2.0/2.1/2.2 already scheduled up to Eclipse 3.8

- Stable version available since version 10.x

- Usable version with some features available since version 7.0

Always keep in mind

- IDE integration
  - Sometimes still in an early stage
  - More updates in the future
- Usage concept
  - (totally) different from CVS/ SVN
- Build server integration
  - A plug-in for Git is required
    - Available for Hudson and Jenkins
- No central server
  - Makes backup of latest version more difficult

The first step is always the hardest

- Create a new **branch** for every feature item, bug fix, …
- **commit** as often as you like
  - **push** once when the feature, bug fix, … is complete
- **reset** (revert) depends on where the changes are
  - Command line
    - **git checkout** *file* for not staged (not added) files
    - **git reset HEAD** *file* for staged files
  - EGit requires simple selection of reset type (soft, mixed, hard)
- SHA-1 hash value instead of a revision number
  - Usually the first six or seven characters are enough

(E)Git Pros and Cons

Performance: extremely fast even in large projects
Offline mode: no server connection required
Branching/ merging: fast merging is done all the time
Fully distributed: no central server required
Repository size: requires less space as SVN
Search view: search for commits in Eclipse
Creativity: experimental branches for new ideas

Revisions: Hash value required for distributed versioning
No partial checkout: clones the entire repository

And the winner is…

- **EGit is ready**
  - Use it for your next new project
  - Faster and much more fun
  - Some commands are not available in EGit yet
    - Install command line as well
    - Updates already scheduled until Eclipse 3.8

- As a temporary alternative
  - Connect your existing repositories via
    - **git svn**
    - **git cvsimport**

Keep in mind

Once started, it is very difficult to go back…

More information

- Git http://git-scm.com
- Git Community Book http://book.git-scm.com/
- ProGit http://progit.org
- Git Cheat Sheet http://ktown.kde.org/~zrusin/git/
- GitHub www.github.com
- Eclipse JGit www.eclipse.org/jgit
- Eclipse EGit www.eclipse.org/egit
- Linus Torvalds on Git http://www.youtube.com/watch?v=4XpnKHJAok8
- It's time to stop using Subversion http://altdevblogaday.org/2011/03/09/its-time-to-stop-using-subversion

Thank you!