Version Control

Michal Sojka, Martin Molnár

sojkam1 (at) fel.cvut.cz
molnam1 (at) fel.cvut.cz

2. 12. 2015

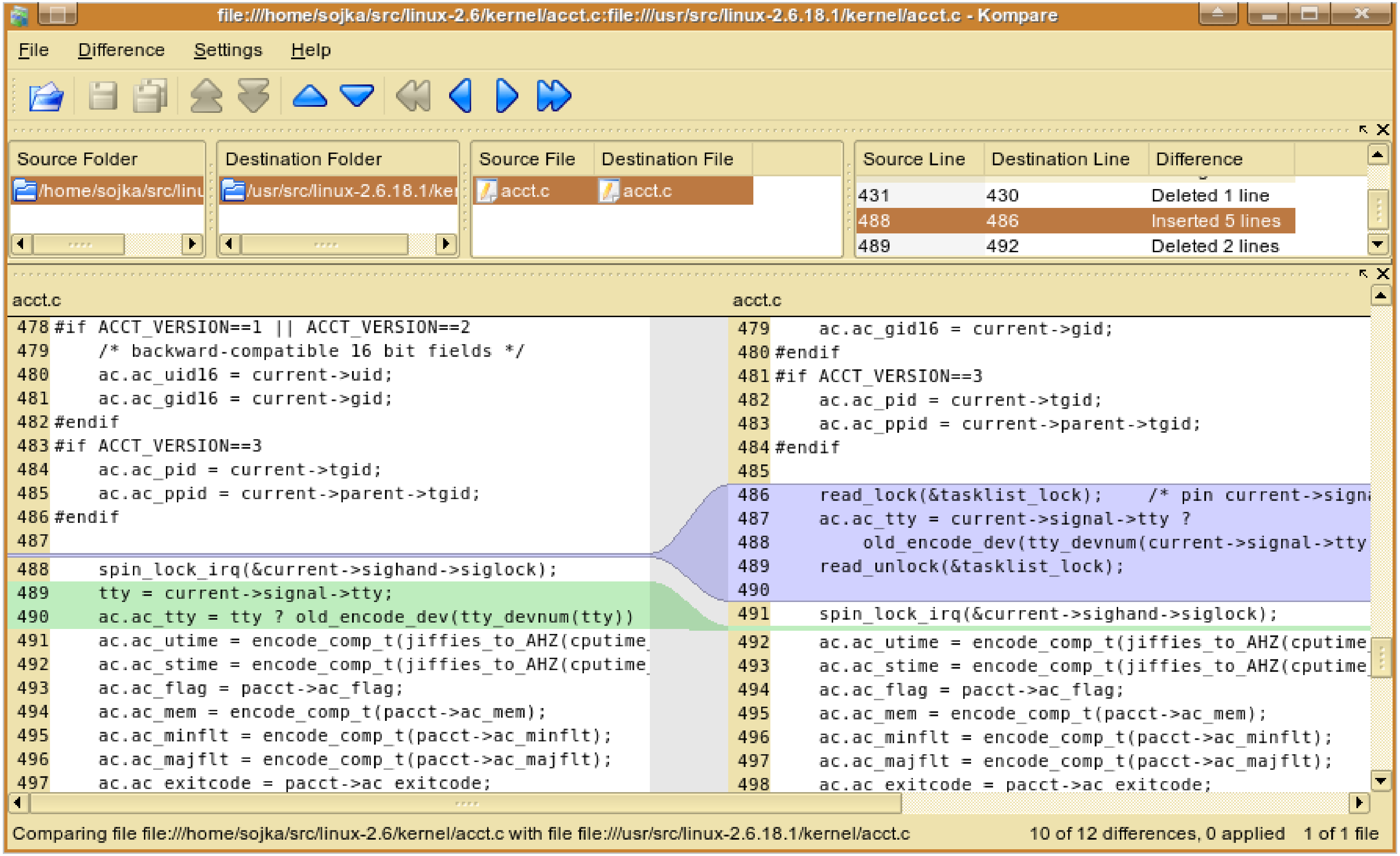
What is version control good for?

- More people work together on one (software) project.
- One person works with multiple computer (home, office).
- What is the last version on my file?
- Who did the change which caused the bug?
- How did I correct that error last year?
- What are the changes since the last release?

Available solutions

- Manual comparison of files
  - commands: **diff** and **patch**
  - Graphical diffs (kdiff3, WinMerge)
- Version control systems (software)
  - local: RCS
  - networked
    - Centralized repository: CVS, Subversion, ...
    - Distributed repository: Darcs, git, Monotone, Bitkeeper, bzr, Mercurial...

Manual file comparison


file:///home/sojka/src/linux-2.6/kernel/acct.c:file:///usr/src/linux-2.6.18.1/kernel/acct.c - Kompare
File
Difference
Settings
Help

Source Folder	Destination Folder	Source File	Destination File
/home/sojka/src/linu	/usr/src/linux-2.6.18.1/ker	acct.c	acct.c

Source Line	Destination Line	Difference
431	430	Deleted 1 line
488	486	Inserted 5 lines
489	492	Deleted 2 lines

acct.c

```
478 #if ACCT_VERSION==1 || ACCT_VERSION==2
479     /* backward-compatible 16 bit fields */
480     ac.ac_uid16 = current->uid;
481     ac.ac_gid16 = current->gid;
482 #endif
483 #if ACCT_VERSION==3
484     ac.ac_pid = current->tgid;
485     ac.ac_ppid = current->parent->tgid;
486 #endif
487
488     spin_lock_irq(&current->sighand->siglock);
489     tty = current->signal->tty;
490     ac.ac_tty = tty ? old_encode_dev(tty_devnum(tty))
491     ac.ac_utime = encode_comp_t(jiffies_to_AHZ(cputime
492     ac.ac_stime = encode_comp_t(jiffies_to_AHZ(cputime
493     ac.ac_flag = pacct->ac_flag;
494     ac.ac_mem = encode_comp_t(pacct->ac_mem);
495     ac.ac_minflt = encode_comp_t(pacct->ac_minflt);
496     ac.ac_majflt = encode_comp_t(pacct->ac_majflt);
497     ac.ac_exitcode = pacct->ac_exitcode;
```

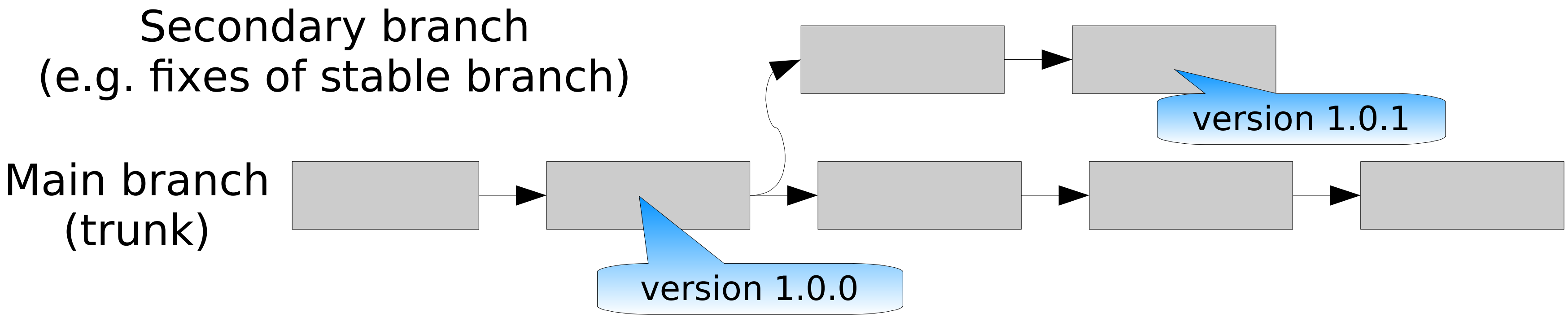
acct.c

```
479     ac.ac_gid16 = current->gid;
480 #endif
481 #if ACCT_VERSION==3
482     ac.ac_pid = current->tgid;
483     ac.ac_ppid = current->parent->tgid;
484 #endif
485
486     read_lock(&tasklist_lock);    /* pin current->signa
487     ac.ac_tty = current->signal->tty ?
488         old_encode_dev(tty_devnum(current->signal->tty
489     read_unlock(&tasklist_lock);
490
491     spin_lock_irq(&current->sighand->siglock);
492     ac.ac_utime = encode_comp_t(jiffies_to_AHZ(cputime
493     ac.ac_stime = encode_comp_t(jiffies_to_AHZ(cputime
494     ac.ac_flag = pacct->ac_flag;
495     ac.ac_mem = encode_comp_t(pacct->ac_mem);
496     ac.ac_minflt = encode_comp_t(pacct->ac_minflt);
497     ac.ac_majflt = encode_comp_t(pacct->ac_majflt);
498     ac.ac_exitcode = pacct->ac_exitcode;
```

Comparing file file:///home/sojka/src/linux-2.6/kernel/acct.c with file file:///usr/src/linux-2.6.18.1/kernel/acct.c   10 of 12 differences, 0 applied   1 of 1 file

Version control systems

- Store the whole project history
- Allow for commenting individual changes
- Store when and who did the change
- **Merge** changes in the same file from multiple people
- Allows for multiple development **branches**
- Can **tag** some revisions by a symblic name



VCSs with centralized repository

- CVS, Subversion (SVN)
- There is only one repository usually stored on a server
- Every developer has a *working copy* in his computer
- Basic operations:
  - After there is something changed in the working copy, a new revision is stored in the repository (commit, check-in)
  - Update the working copy from repository (update, check-out) Local (uncommited) changes are merged with updates
- Terminology:
  - HEAD – the latest revision in the repository
  - BASE – revision which was checked-out (after checkout BASE=HEAD until somebody commits to the repository)

Recommendations for commit messages

- Use the whole sentences (with verb). For example:

  Fixed bug in ...
  Added computation of PI.

  Wrong:

  Computation of pi.

- It must be clear what was changed only by looking at the message (not at the code).

- For longer message start with a brief one line description and continue with additional paragraphs.

- Many open-source projects require “Certificate of Origin”:
  - Signed-off-by: Name <email@example.com>

Recommendations for commit messages (cont.)

- Ideal commit message answers the following questions:
  - Why did you change that code?
  - What led you to that code (motivation, problem report, use-case, etc.)?
  - What options did you consider?
  - Why did you select the option taken out of those?
  - What is the intended result?
  - How much testing was done?

VCSs with distributed repository

- Git, Darcs, Monotone, bzr, mercurial, BitKeeper
- No need for centralized repository (but any repository can be used as a central one)
- Working copy is also repository at the same time
- Changes in the working copy are first recorded (**committed**) to the “local” repository.
- Then changes can be sent to other developer's repositories or to the central repository (**push**).
- Changes can also be **pulled** from other (central) repositories.
- Advantages:
  - You can work off-line
  - Possibility of having multiple versions (branches) of projects and move changes between them

What is git?

- Source control management (SCM) system designed for sharing large amounts of source code among a distributed group of developers
- Initially written by Linus Torvalds to manage Linux kernel sources
- simply and concisely: **git is a stupid (but extremely fast) directory content manager**
- Drawbacks (not completely true today)
  - Steeper learning curve (27 high-level commands, 140 in total)
  - Windows support not so mature
    - people continuously improve it
- Homepage: http://git-scm.com - contains useful information (manual, tutorials, wiki, etc.) about git

What does GIT stands for?

- According to *man git*, "git" can mean anything, depending on your mood:
  - random three-letter combination that is pronounceable, and not actually used by any common UNIX command. The fact that it is a mispronunciation of "get" may or may not be relevant.
  - stupid. contemptible and despicable. simple. Take your pick from the dictionary of slang.
  - "global information tracker": you’re in a good mood, and it actually works for you. Angels sing, and a light suddenly fills the room.
  - "goddamn idiotic truckload of sh*t": when it breaks

Main features

- **fully distributed** – no need for central repository (this is a good thing, why?). Changes are committed to the local (cloned) repository
- **fully peer-to-peer**
  - repository can be based on one or more remote repositories
  - repository can be published for other developers to use
- **complex merges**
  - different merge algorithms – starting with a very fast stupid one progressing to more complex and time consuming ones
  - able to recognize and handle duplicate changes
  - If the merge cannot be done automatically, git gives you a powerful tool to help you with the merge.
- **file content tracking** – does not record only file content changes but whole file content

Features

- Very efficient storage of history:
  - Unpacked Linux 2.6.32 sources:
    - du -ch `git ls-files`: **410 MB**
  - 4.5 years of Linux development history = 186 thousands of commits = 113 commits every day (in average)
    - du -sh .git: **419 MB**
  - Unpacked Linux 4.3: 688 MB
  - 10.5 years history, 548 thousands commits (152 commits daily): 1124 MB

git components

- **Object Database**
  - collects objects of four types: blob, tree, commit, tag
  - objects are addressed by SHA1 hash of their content
- **Index**
  - current tree **cache**
  - stores the next revision to be committed

Object Database I.

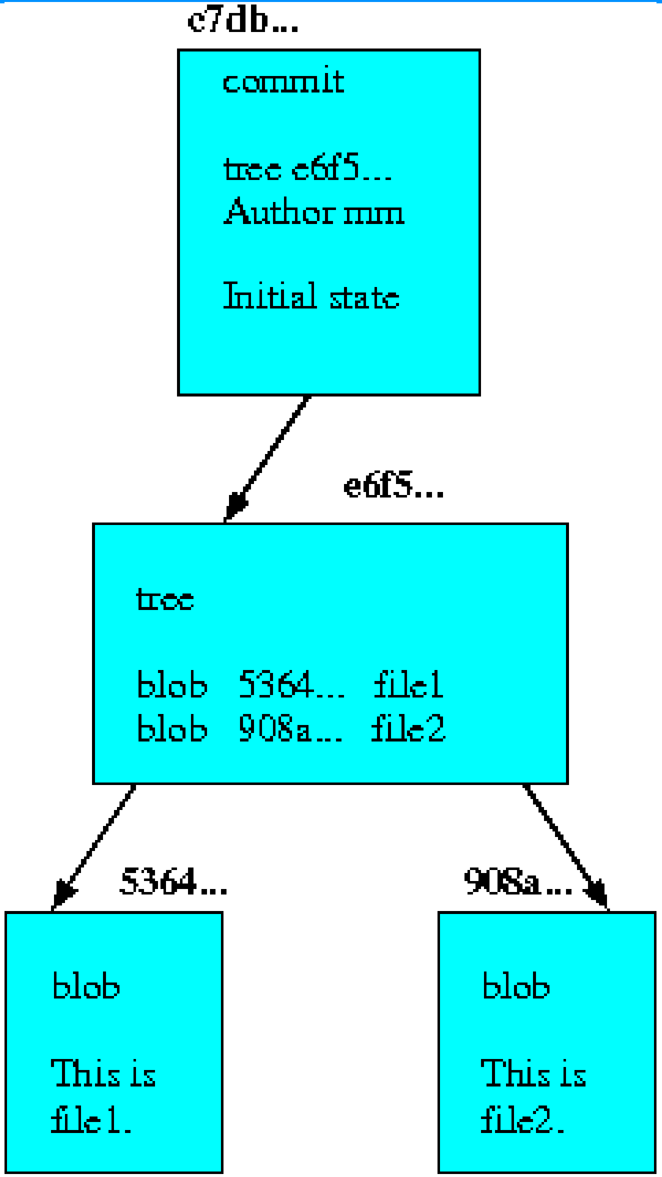
- **blob object**
  - represents **contents** = one version of a file
  - if two files in a directory tree (or in multiple different versions of the repository) have the same contents, they will share the same blob object
- **tree object**
  - represents one directory
  - contains sorted list of text lines with the following information: *mode,object type, SHA1, path name*
  - information about blobs and tree objects lying in the directory
  - several tree objects forms hierarchical directory **structure**

Object Database II.

- **commit object**
  - contains by the reference to related tree object, the parent commits, commentary
  - sequence of commit objects provides the history
  - commit objects tie the directory structures together into a acyclic graph (DAG)
- **tag object**
  - assigns symbolic name to particular object reference e.g. commit object associated with a named release
  - contains SHA1, object type, symbolic name of referenced object and optionally a signature

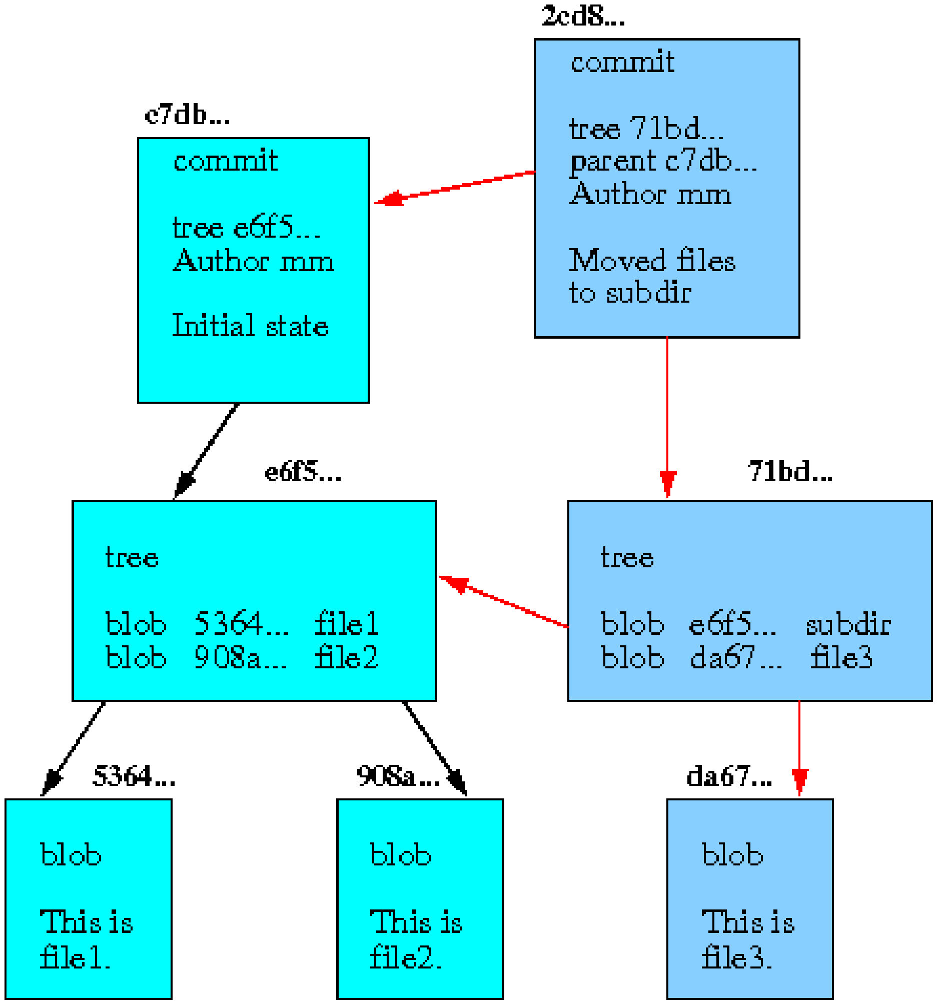
Example: Object Database I.

- 1. Start with a new repository
- 2. Create file1 with the content: “This is file1.”
- 3. Create file2 with the content: “This is file2.”
- 4. Update the index
- 5. Make an initial commit



Example: Object Database II.

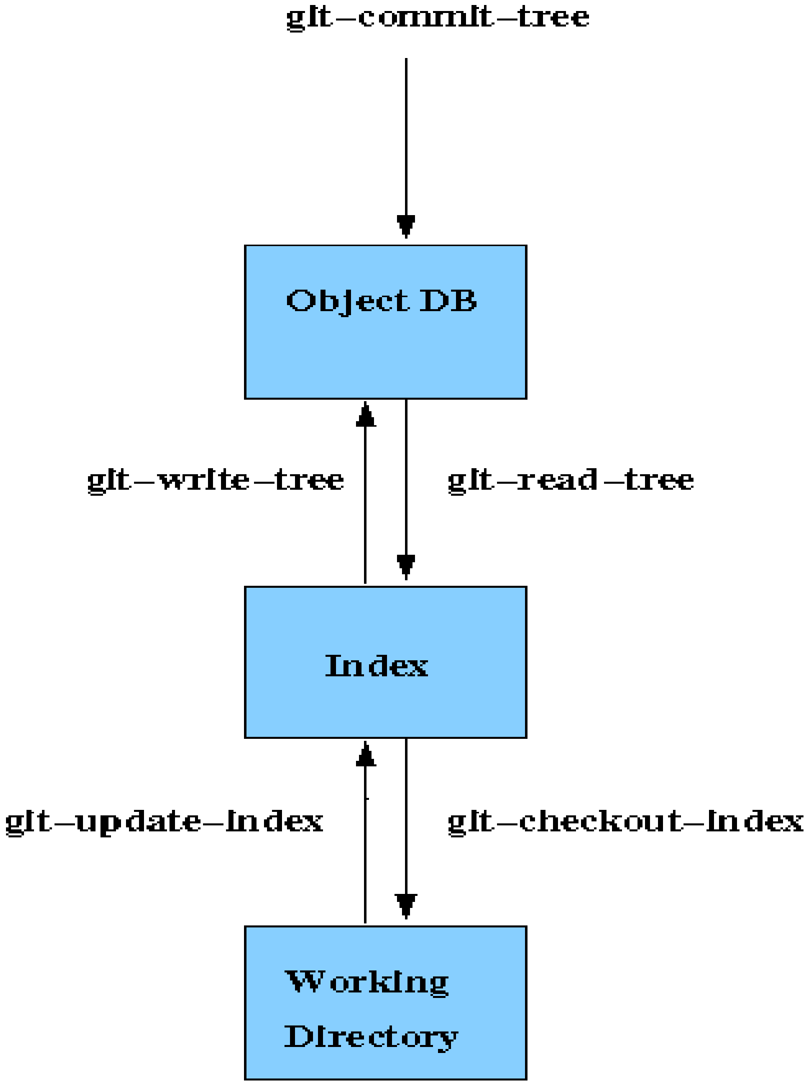
- 1. Move file1 and file2 into subdirectory
- 2. At top level, create file3 with the content: “This is file3.”
- 3. Update the index
- 4. Make a commit



Index

- simple binary file, which contains an efficient representation of a virtual directory content
- it is implemented by a simple array that associates a set of names, dates, permissions and content (blob) objects together
- serves as staging area to prepare commits
- helps with merge conflicts resolving
- improves performance (speed of operations)

Internal git workflow



Plumbings and Porcelains

- **plumbings** are low level git commands e.g. git-write-tree, git-commit-tree, etc.
- **porcelains** are high level git commands (e.g. git commit calls git-write-tree and git-commit-tree) and other frontends:
  - git gui, gitk, qgit – graphical tools
  - tig – text mode git browser
  - Gitweb, Cgit
  - TopGit, StGit – simplifies patch-queue management in git

Example

- mkdir git-test; cd git-test
- **git init** – initialize empty git repository in the current working directory
- echo “Hello world” > hello
- **git add** hello – adds file to the Index
- **git status** – shows the state of the index
- **git commit** -m “Adding file hello.” – commits changes

Example: Linux git repository

- **gitclone \** git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
- cd linux ; **git pull –** pull new revisions from remote repository
- **git gc** – clean and compress object database (garbage collect)

Projects using Git

- Linux kernel
- LibreOffice (OpenOffice)
- GNOME
- KDE
- Perl
- Qt
- Android
- PostgreSQL
- Fedora
- Debian
- X.org

References

- *“How To Git It”, ”Embracing the Git Index” ,“Collaborating With Git”* Jon Loeliger, Linux Magazine, March 2006
- Git man pages
- Tutorials at http://git.or.cz

What is quilt?

- tool to manage large sets of patches by keeping track of the changes each patch makes
- patches are applied incrementally on top of the base tree plus all preceding applied patches=> stack of patches
- quilt is command-line tool invoked by: quilt *command*

Some Quilt commands

- **new** *patchname* - create a new patch and insert it after the topmost patch in the patch series file
- **add** *filename* - Add file to the topmost patch
- **refresh** – refreshes the topmost patch
- **push** – apply patches from the series file
- **pop** - remove patch(es) from the stack of applied patches

Example I.

- mkdir quilt-test
- echo Hello > a.txt
- **quilt new** a.patch
- **quilt add** a.c
  - track a.c in the a.patch
  - the content of a.c is backuped in *.pc/a.patch* directory
- modify a.c
- **quilt refresh**
  - updates/creates a.patch in the *patches* directory
- create file b.c
- quilt new b.patch – b.patch is now on the top of patch stack