Git – A gentle introduction

–FIXME Hackerspace Lausanne–

esc

2012-09-26

Outline

Designed as an interactive talk, please interrupt to ask questions.

Advantages and Disadvantages of Distributed Version Control Systems (DVCS)

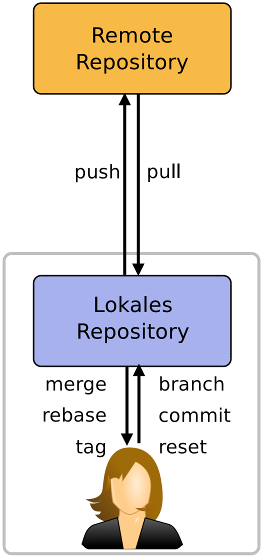
Advantages:

- Every developer has a complete copy of the public history
  - Enables working offline
  - Commands much faster
  - Implicit protection against manipulation
- No «single point of failure»
  - Server offline, disgruntled developer, security breach . . .
- Allows any workflow
  - For enterprises: centralised workflow

Disadvantages:

- Lots of freedom, appropriate policies must be established
- Slightly more complex setup

Autonomy of local Repositories



- *Remote* and local repository are not that different
- Exchange between repositories via push/pull
  - *Push*: Upload own changes
  - *Pull*: Download other peoples changes
- Everything else happens only locally

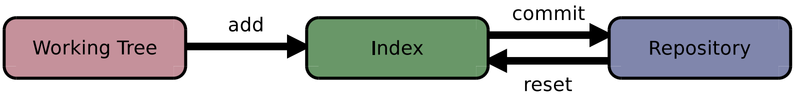
Git Special Fetaures

Git has a number of special features, understanding them is paramount to effective use

- Staging Area
- Object Model
- Commit Graph

Staging Area

- Git allows you to assemble a commit incrementally
- First add hunks to the stage, then commit everything in the stage



- Has many names: *stage*, *index*, *cache*.
- Very relevant for everyday work

Object Model

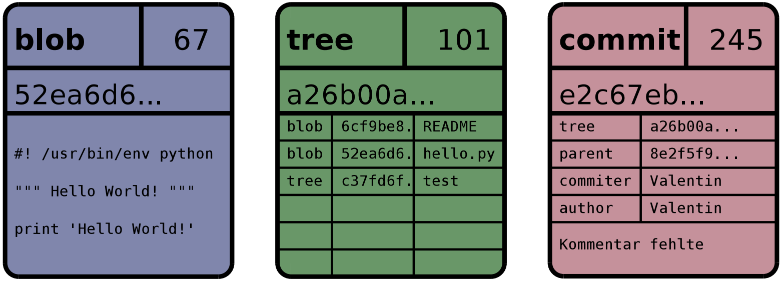
Imagine we wish to track the following repository:

```
/
├── hello.py
├── README
└── test/
    └── test.sh
```

http://krzz.de/3u

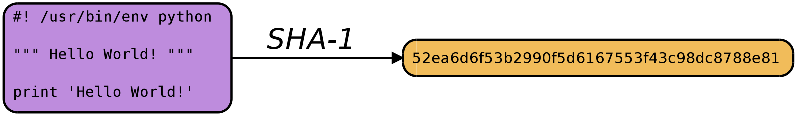
Git-Objects

- *Blob*: contents of a file
- *Tree*: collection of blobs and other trees
- *Commit*: reference to a tree and metadata
  - *Author* and *Commiter*, *Parents*, *Commit-Message*, etc..



Content Addressable System

- Each object has a unique identifier, its SHA-1
- ... is Calculated from its content
- ... allows retrieval using this identifier



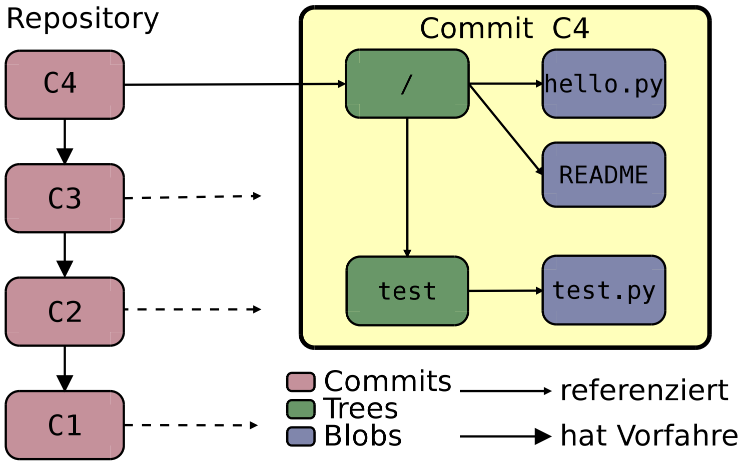
```
#! /usr/bin/env python

""" Hello World! """

print 'Hello World!'
```

- Importantly each *Commit* object contains the SHA-1 of its parent(s)
- Objects stored in an object storage (files on disk)
- Implements de-duplication

Building History from Objects



Object Database

```
$ git cat-file commit e2c67ebb6d2db2aab831f477306baa44036af635
tree a26b00aaef1492c697fd2f5a0593663ce07006bf
parent 8e2f5f996373b900bd4e54c3aefc08ae44d0aac2
author Valentin Haenel <valentin.haenel@gmx.de> 1294515058 +0100
committer Valentin Haenel <valentin.haenel@gmx.de> 1294516312 +0100

Kommentar fehlte
```

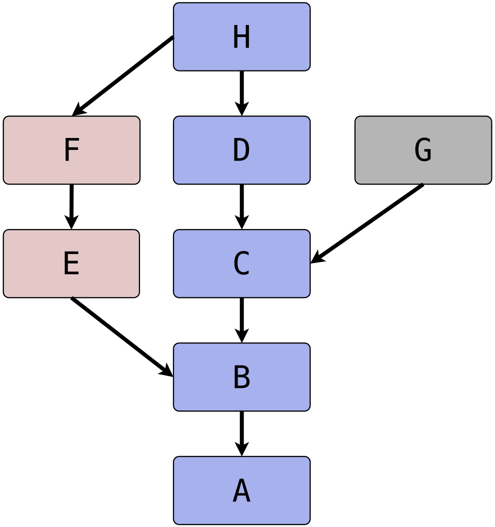
```
$ git ls-tree a26b00aaef1492c697fd2f5a0593663ce07006bf
100644 blob 6cf9be8017a937ca9f442290bcc8b2db13f12ab4    README
100644 blob 52ea6d6f53b2990f5d6167553f43c98dc8788e81    hello.py
040000 tree c37fd6f7d4f9619448f0feafec09ef5d18b58712    test
```

```
$ git cat-file blob 52ea6d6f53b2990f5d6167553f43c98dc8788e81
#!  /usr/bin/env python

""" Hello World!  """

print 'Hello World!'
```

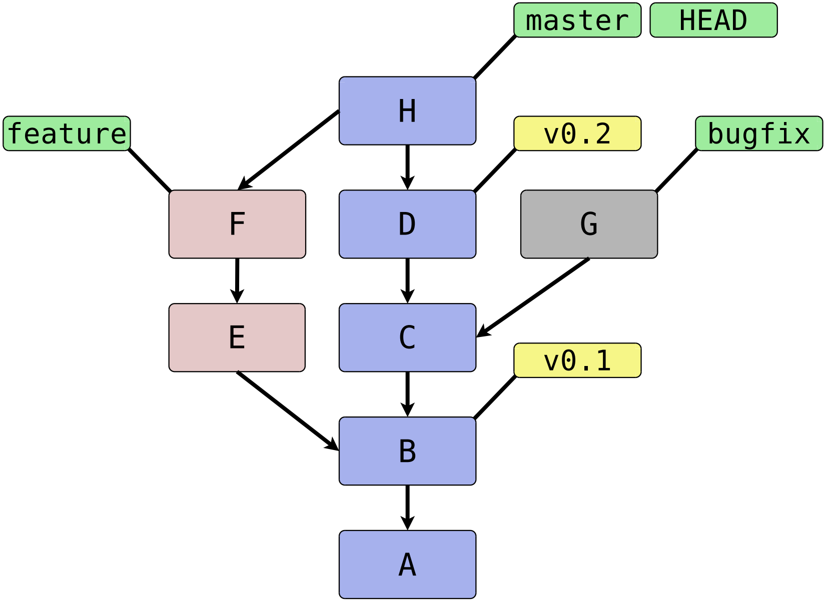
Commit Graph



So What about Branches

- Now that we have an implicit graph structure, branches become obvious
- Pointers into the commit-graph
- Updated when new commits are added
- Tags are like branches but don't get updated

Commit Graph



Cheap and Effective Branching

- Absolutely git's killer feature!
- The problem isn't the branching, it's the merging
- Merge basis derived from graph
- An intermediate user will create (and merge!) multiple branches on one day
- Branches are mostly local, no one sees your mess

Visualization

- Since the history is a graph, it's important to visualise it
- There are many tools for this job

- Gitk
- Gitx

- Also: tig, git-big-picture, gitg, qgit, ...
- See the git-wiki for more open-source and commercial alternatives

Summary

- Git commands manipulate the graph structure
  - Create new bifurcations
  - Add/Move/Remove nodes

Questions?

Introduction

- The basic interface to git is the command-line
- Git uses *subcommands*
- There are *porcelain* and *plumbing* commands
- GUIs exist and are quite functional (so I've been told)

- Will «whizz» through the list of most important commands
- Shout if you are unfamiliar with these or want them demonstrated

Initialisation

- git init
- git clone

Configuration and Information

- git config
- git status
- git help

Index and Commits

- git add
- git commit
- git reset

Branch manipulation

- git checkout
- git branch
- git merge

Inspection

- git log
- git diff

Remote Interaction

- git remote
- git fetch
- git pull
- git push

Extended

- git mv
- git rm
- git alias
- git bisect
- git revert
- git shortlog
- git reflog
- git cherry-pick
- git format-patch
- git send-email
- git am

… And More

- git describe
- git tag
- git grep
- git stash
- git submodule
- git subtree
- git show
- git rebase
- git filter-branch
- git gc
- git svn

… and so on … (around 200 commands)

Common Pitfalls (1)

Forgetting to `git push` and dropping your laptop

- Work may be lost on a broken hard drive
- Push at regular intervals
- Don't drop your laptop

Incorrect use of `git reset --hard`

- Think before you use `--hard`
- Just try not to use it all the time

Common Pitfalls (2)

Deleting `stash` by accident

- The last paragraph in the `git stash` manpage shows how to find deleted stashes

Incorrect use of `git clean -dfx`

- Removal of unstaged files

Common Pitfalls (3)

- Any history rewriting `git rebase` and `git filter-branch`
- Also try: `git reflog`
- See also: `http://gael-varoquaux.info/blog/?p=137`

Famous Last Words

- You don't need to know all the commands
- Once you understand concepts, the commands make sense
- The learning curve might be a bit steep
- There are lots of books/websites/forums/chats etc.
- Main entry-point: http://git-scm.com/

Good Luck!