

git it, got it? Good!

what is git?

git is a fast, content-addressable, decentralized and symmetrically ad hoc synchronizable, cryptographically verified filesystem, stored as a directed acyclic graph of commits, trees, and blobs, with SHA-1 pointers as edges, backed by a POSIX-compatible filesystem, with both a simple storage format and a space- and seek-efficient, compressed, delta chained pack format.

the end

appendices

a brief history of ~~time~~ git

- ~ development began on April 3, 2005
 - ~ stated reason: “everything else sucks”
- ~ git was announced on April 6, 2005
- ~ git became self-hosting on April 7, 2005
- ~ git has managed the linux kernel since June 16, 2005

getting git

source, packages, git git, configuration

getting git: source

```
curl -O http://www.kernel.org/pub/software/scm/git/  
git-1.5.4.tar.gz
```

```
tar xzvf git-1.5.4.tar.gz
```

```
cd git-1.5.4
```

```
make prefix=/usr/local
```

```
sudo make prefix=/usr/local install
```

getting git: packages

~ Leopard

```
sudo port install git-core +svn
```

~ Debian/Ubuntu

```
sudo apt-get install git-core
```

~ Fedora

```
yum install git-core
```

getting git: git git

```
git clone git://git.kernel.org/pub/scm/git/git.git
```

configure some stuff

```
# set your name and email, used for commit messages  
git config --global user.name "Your Name Comes Here"  
git config --global user.email you@yourdomain.example.com
```

```
# enable nice colorful output  
git config --global color.diff auto  
git config --global color.status auto  
git config --global color.branch auto
```

how git works

structure & behavior

git structure

- ~ content addressable
- ~ commits, trees and blobs
- ~ the history graph
- ~ references
- ~ HEAD
- ~ index
- ~ .git

content addressable

(or: the mighty SHA1)

- ~ git stores content as objects
- ~ git calculates SHA1 hash of content
- ~ the object is addressed by its hash
- ~ SHA1 hashes are universally unique
- ~ duplicate content is stored only once

commits, trees and blobs

(or: the mighty SHA1)

~ blobs store the contents of files



commits, trees and blobs

(or: the mighty SHA1)

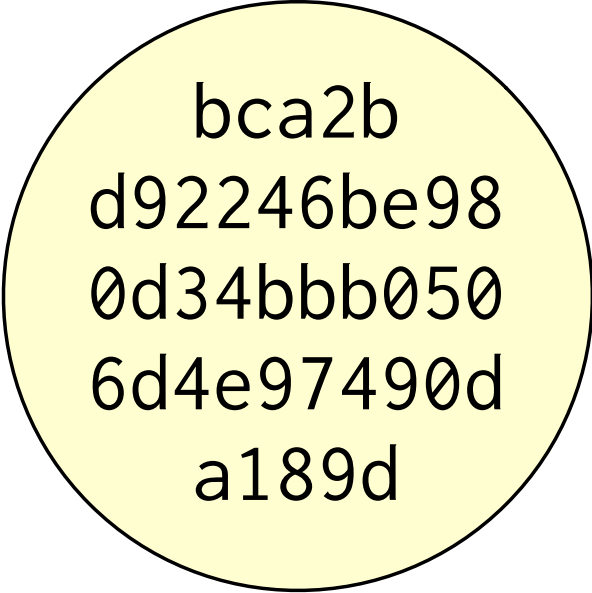
- ~ blobs store the contents of files
- ~ trees store trees and blobs



commits, trees and blobs

(or: the mighty SHA1)

- ~ blobs store the contents of files
- ~ trees store trees and blobs
- ~ commits store a tree of trees and blobs

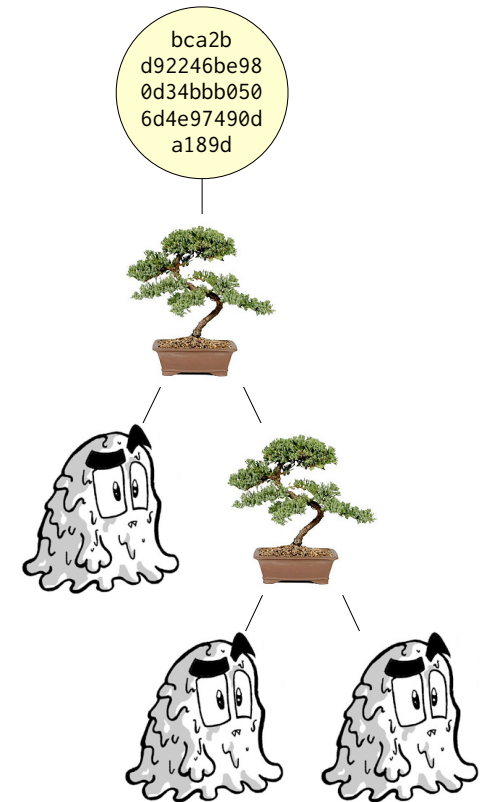


bca2b
d92246be98
0d34bbb050
6d4e97490d
a189d

commits, trees and blobs

(or: the mighty SHA1)

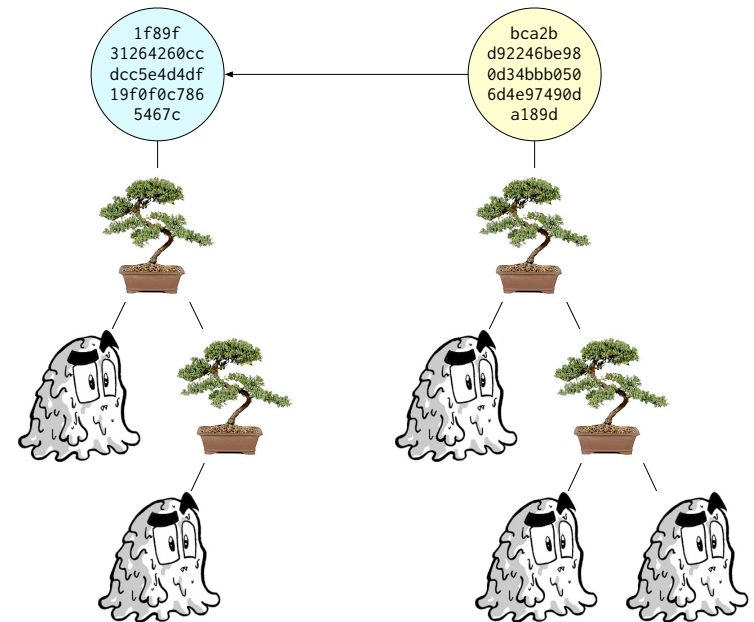
- ~ blobs store the contents of files
- ~ trees store trees and blobs
- ~ commits store a tree of trees and blobs



commits, trees and blobs

(or: the mighty SHA1)

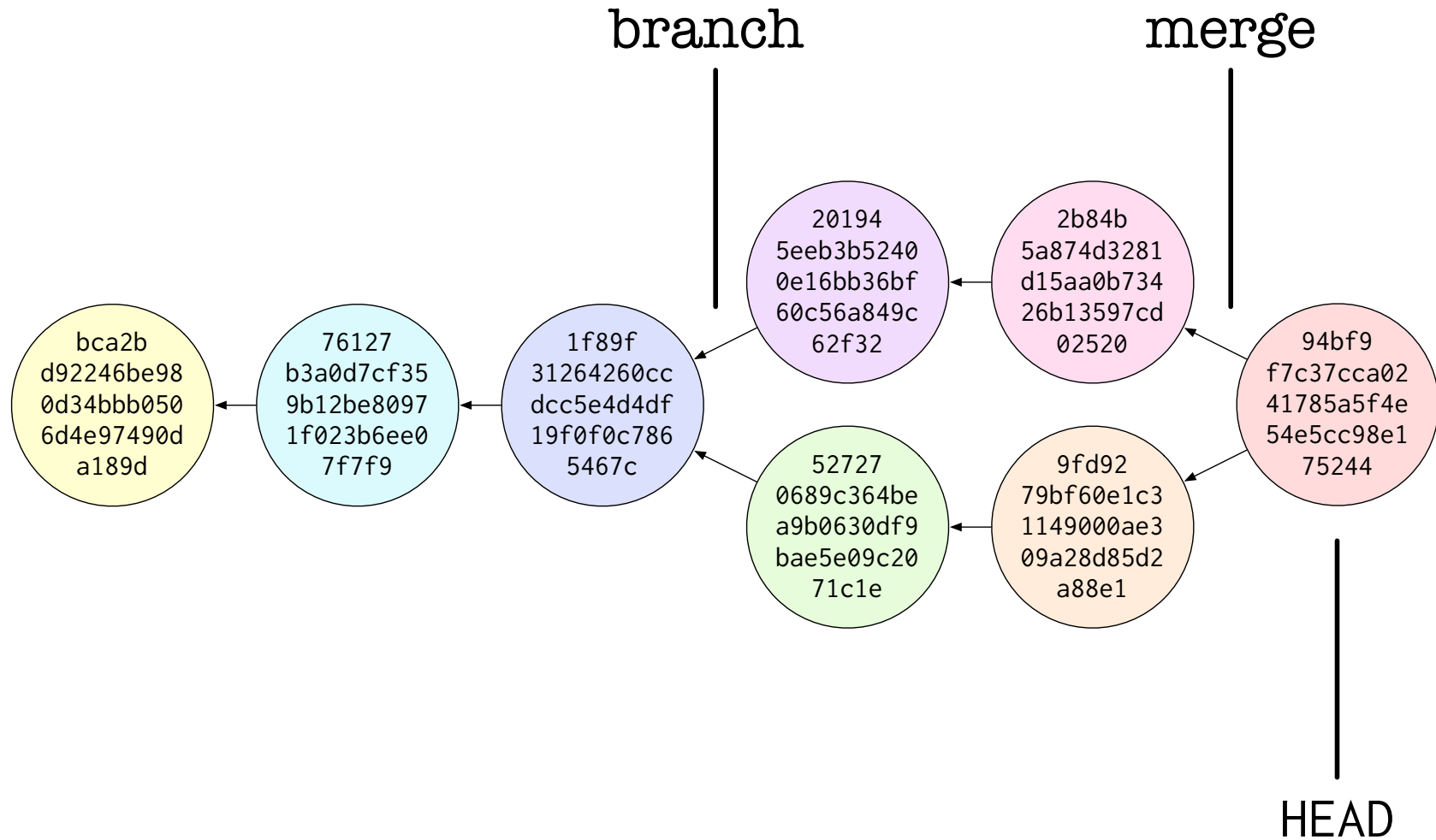
- ~ blobs store the contents of files
- ~ trees store trees and blobs
- ~ commits store a tree of trees and blobs
- ~ commits point to their parent(s)



the history graph

- ~ commits point to 0 or more parents
- ~ these commits and their parents form a directed acyclic graph
- ~ or, in pretty pictures:

the history graph



references

- ~ heads (branch tips)
- ~ remotes (upstream, downstream)
- ~ tags
- ~ stored in `.git/refs/`*

HEAD

- ~ reference to current branch
- ~ stored in `.git/HEAD`

index

(the index, the index, what, what, the index)

- ~ a stored version of your working tree.
- ~ determines what is committed

.git

- config
- refs/
 - heads/*
 - remotes/*
 - tags/*
- objects/*
- index
- HEAD

git behavior

- ~ distribution
- ~ making changes
- ~ branching and merging
- ~ checking out
- ~ fetching
- ~ pushing and pulling
- ~ rebasing

distribution

- ~ work locally, commit locally
- ~ work offline
- ~ redundancy
- ~ scalability

making changes

~ `git add`

~ adds new or modified files to the index

~ `git commit`

~ adds new or modified files to the index

~ `git commit -a`

~ automatically commit new or modified files

branching

- ~ `git branch <name>`
- ~ **creates a new reference from HEAD**
 - ~ `git` creates a new file at `.git/refs/heads/<name>`
 - ~ file contains the SHA1 of the branch point

merging

- ~ `git merge <name>`
- ~ merges `<name>` into HEAD
 - ~ git performs a three-way merge (by default)
 - ~ file contains the SHA1 of the branch point
- ~ fast-forward
 - ~ a fast-forward merge occurs when the history of `<name>` already includes the HEAD commit
 - ~ the current branch's reference is updated to the end point of `<name>`
 - ~ a new commit is not created

checking out

- ~ `git checkout <name>`
- ~ switches HEAD and the working copy
 - ~ HEAD is changed to <name>
 - ~ file contains the SHA1 of the branch point
- ~ `git checkout -b <name>`
 - ~ a new branch <name> is created
 - ~ git then checks this branch out

fetching

- ~ `git fetch <repository> <name>`
- ~ downloads objects and references from `<repository>`
 - ~ the downloaded references are stored in `git/FETCH_HEAD`
 - ~ these can then be checked out or merged

pushing

- ~ `git push [<repository> <name>]`
- ~ updates the references at the remote `<repository>`, or specifically the reference `<name>`

pulling

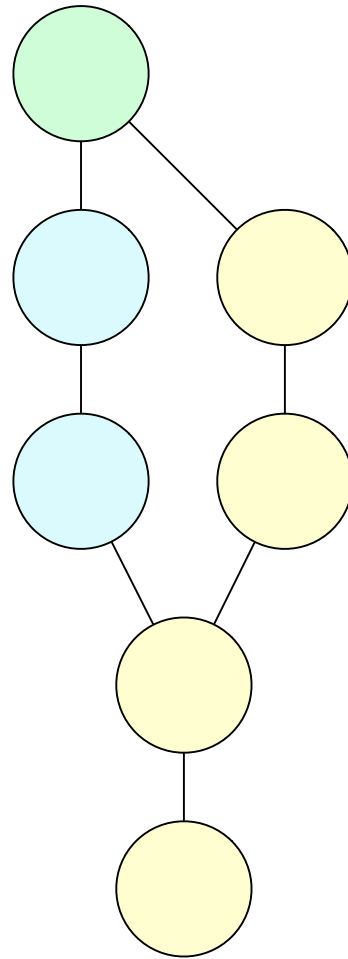
- ~ `git pull <repository> <name>`
- ~ runs `git fetch` and then `git merge`, merging into HEAD

rebasing

- ~ `git rebase <name>`
- ~ takes local changes not present in `<name>`, stores them temporarily as patches, updates the local reference to the head of `<name>`, and applies the stored patches.

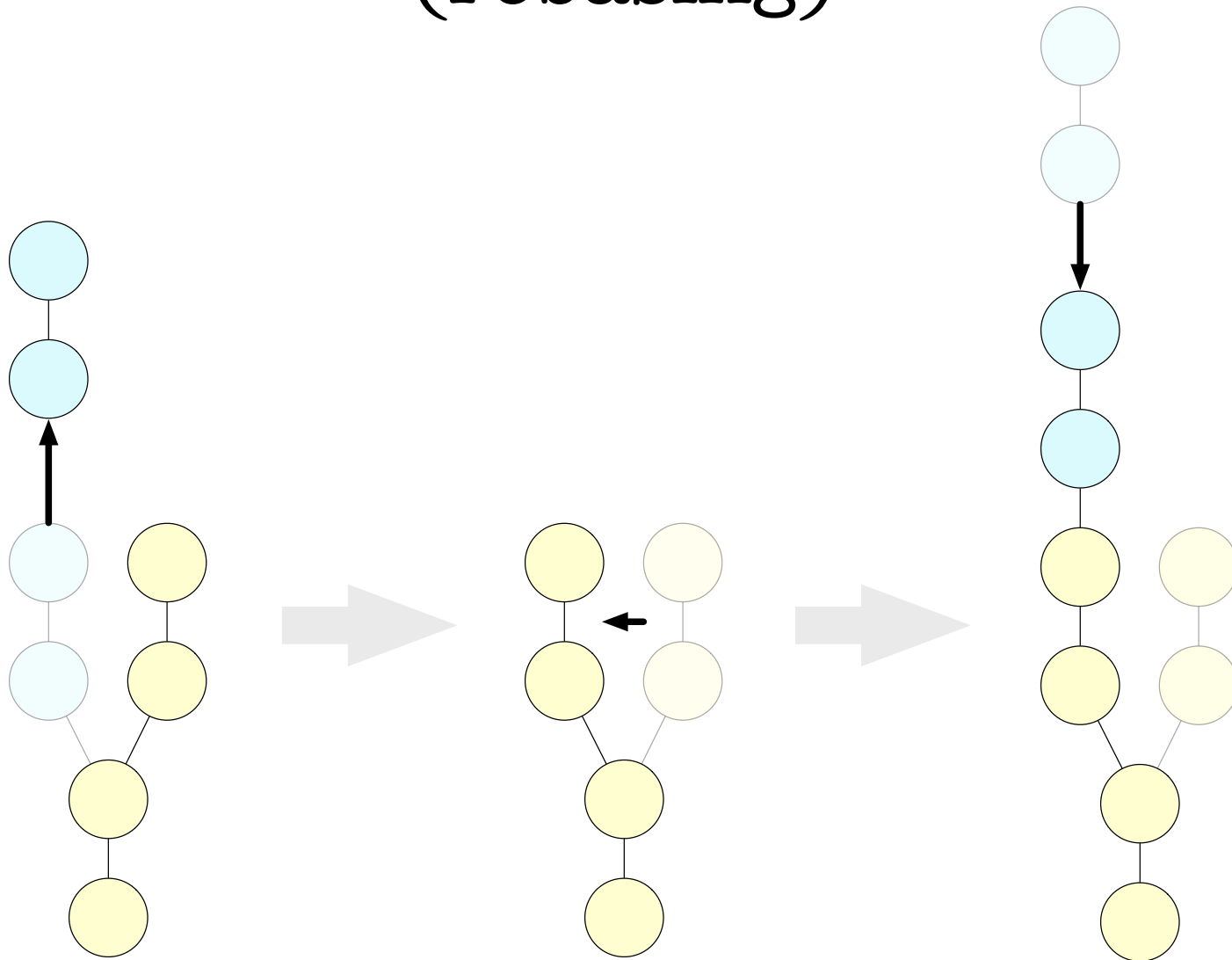
rebasing vs merging

(merging)



rebasing vs merging

(rebasing)



how to use git

play with yourself – and others!

playing with yourself

- ~ init a git repository
- ~ add local files
- ~ check status
- ~ your first commit
- ~ logging and diffing
- ~ branching and merging

init a git repository

```
git init .
```

add local files

```
git add .
```

check status

```
git status .
```

your first commit

```
git commit
```

logging and diffing

```
git log
```

```
git diff
```

logging

```
# Show the whole commit history in oneline format  
git log --pretty=oneline
```

```
# Show the changes during the last two weeks  
git log --since="2 weeks ago"
```

```
# Show the commits that are in the "test" branch but not yet  
# in the "release" branch.  
git log release..test
```

diffing

```
# Changes in the working tree not yet staged  
# for the next commit.  
git diff
```

```
# Changes between the index and your last commit  
# (what you would be committing if you run "git commit"  
# without "-a" option.)  
git diff --cached
```

```
# Changes in the working tree since your last commit  
# what you would be committing if you run "git commit -a"  
git diff HEAD
```

diffing (cont.)

(helpful git aliases for your .bashrc)

```
alias staged="git diff --cached"
```

```
alias unstaged="git diff"
```

```
alias both="git diff HEAD"
```

branching and merging

`git branch`

`git merge`

branching

```
# Create a local topic branch to work in  
git branch my-work
```

```
# Switch to new local branch  
git checkout my-work
```

```
# Create and switch to new local branch in one step  
git checkout -b my-work
```

branching (cont.)

```
# Switch back to master branch  
git checkout master
```

```
# Delete unneeded branch  
git branch -d my-old-work
```

merging

```
# Merge my-work branch into current branch  
git merge my-work
```

plays well with others (how to be a team player)

- ~ clone a public git repo
- ~ work in a branch
- ~ stay up-to-date
- ~ merge your changes
- ~ push your changes
- ~ create patches

clone a public repo

```
# Clone from upstream  
git clone git://git.kernel.org/pub/scm/git/git.git
```

work in a branch

(or: never ever work in master)

```
# Create a topical branch for your work  
git checkout -b my-work
```

stay up-to-date

```
# Checkout master branch  
git checkout master
```

```
# Update from upstream  
git pull origin master
```

```
# Rebase your working branch with the new master  
git checkout my-work  
git rebase master
```

merge your changes

```
# Merge changes from your topical branch while in master  
# (this should ideally be a fast-forward merge)  
git merge my-work
```

push your changes

```
# Push your changes to the upstream  
# (while in master branch)  
git push origin master
```

create patches

```
# Create a patch against the upstream  
git diff origin/master > your-patch.patch
```

```
# Create formatted patches for emailing (RECOMMENDED)  
git format-patch origin/master
```

plays well with others (maintaining a public repo)

- ~ create a remote
- ~ track the remote
- ~ merge changes in
- ~ apply patches

create a remote

```
# Add a remote for a downstream repo  
git remote add reindh-dm git://github.com/ReinH/datamapper.git
```

track the remote (the hard way)

```
# Create a local branch to pull from the downstream  
git checkout -b reindh
```

```
# Update the local branch from the downstream  
git pull reindh-dm/master
```

track the remote (the easy way)

```
# Create a local branch that tracks the downstream  
git checkout --track -b reihh reihh-dm/master
```

```
# Update the local branch from the downstream  
git pull
```

```
# NB: --track tells git to remember where to pull from
```

merge changes in

```
# Checkout your master branch  
git checkout master
```

```
# Merge changes in from the local tracking branch  
git merge reindh
```

apply patches

```
# Apply a patch generated by git diff  
git apply patch-name.patch
```

```
# Apply a patch generated by git format-patch  
git am 0001-patch-name.patch
```

```
# NB: Patches generated by git format-patch have additional  
# metadata (author, committer, message, etc) that will be  
# used by git-am when patching.
```

plays well with others

(publish a public repo)

- ~ configure the upstream
- ~ publish your changes
- ~ profit!

configure the upstream

```
# Add and configure the remote for the upstream  
# (git calls the upstream repository "origin")  
git remote add origin git://github.com/sam/dm.git  
git config branch.master.remote origin  
git config branch.master.merge refs/heads/master
```

```
# NB: If your repository is a clone you can skip these steps  
# as the source repository will already be set as origin
```

publish your changes

```
# Publish your changes to the upstream  
git push origin master
```

git hosting

- ~ git-daemon
- ~ gitosis
- ~ Github
- ~ Gitorious

git-daemon

- ~ configure with inet.d or other service manager

git-daemon (inetd)

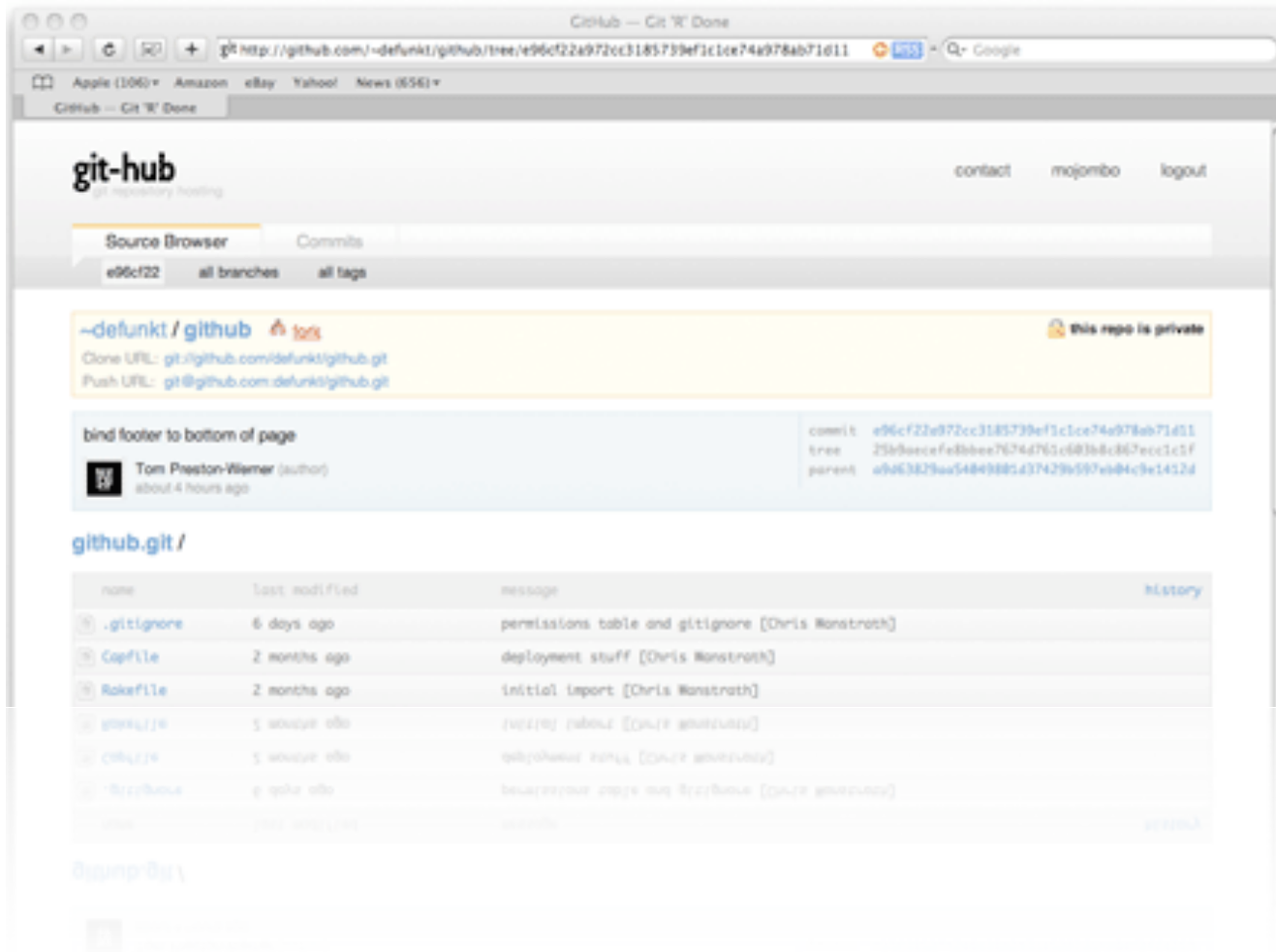
```
/usr/bin/git-daemon git-daemon --inetd --export-all /pub/scm
```

gitosis

- ~ manage multiple public git repositories
- ~ manage access control and ssh keys
- ~ uses git repo to configure
- ~ <http://rubyurl.com/hc4P>

Github

- ~ github.com
- ~ web managed
- ~ currently in beta
- ~ feature rich
- ~ sexy



github.com

Gitorious

- ~ open source git hosting service
- ~ free hosting
- ~ self-hosting
- ~ <http://gitorious.org/>

git resources

- ~ git manual
- ~ git peepcode
- ~ everyday git
- ~ git - svn crash course