IS A STATE OF MIND

Philosophy, under the hood, common and advanced use, modularity, workflows and ecosystem

Nicola Costantino

PHILOSOPHY

Git is a state of mind

GIT: WHAT

- "Git, the stupid content tracker"
- API for DVCS
- Created by Linus Torvalds in 2005 for Linux Kernel source code
- Before: patches & packages, then proprietary BitKeeper

Linus Torvalds

GIT

Linus about the name "*git*" ("*unpleasant person*" in British English slang):

"I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'git'." (source)

```
GIT - the stupid content tracker

"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
   mispronounciation 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

This is a stupid (but extremely fast) directory content manager.  It
doesn't do a whole lot, but what it _does_ do is track directory
contents efficiently.

There are two object abstractions: the "object database", and the
"current directory cache".
```

From the original README (source)

GOD MADE UNIVERSE IN 6 DAYS, LINUS MADE GIT IN 4, GIT SELF-HOSTED IN 1

- Development start: ~April 3, 2005
- Announcement: April 6, 2005
- Self-hosting: April 7, 2005

[prev in list] [next in list] [prev in thread] [next in thread]

List: git
Subject: Re: Trivia: When did git self-host?
From: Linus Torvalds <torvalds () linux-foundation ! org>
Date: 2007-02-27 1:58:57
Message-ID: Pine.LNX.4.64.0702261722310.12485 () woody ! linux-foundation ! org
[Download message RAW]

On Mon, 26 Feb 2007, Linus Torvalds wrote:
>
> The first commit was already self-hosted. It was done manually (write-tree
> etc by hand), but yes, the first commit really is:
>
>   Thu Apr 7 15:13:13 2005 -0700
>
> and the second one (add copyright notices) was done a few minutes later.
>
> So git was self-hosting since April 7, 2005.
>
> Now, exactly when I started git development (ie how long it took before it
> got to that self-hosting stage), I can't remember. I'd say about two
> weeks, probably.

Actually, it must have been less than that.

The first version of git was just ~1300 lines of code, and I have reason
to believe that I started it at or around April 3rd. The reason: I made
the last BK release on that day, and I also remember aiming for having
something usable in two weeks.

And hosting git itself was not that important for me - hosting the kernel
was. And the first kernel commit was April 16 (with the first merges being
a few days later). Which meshes with my "two week goal" recollection.

Not that I'd normally be entirely sure I hit any time goals I set, but I
am pretty sure that I decided that in order to be effective, I didn't
want to do kernel development at the same time as git development, so the
"April 3" date of the 2.6.12-rc2 release is fairly significant.

                        Linus
-
To unsubscribe from this list: send the line "unsubscribe git" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at http://vger.kernel.org/majordomo-info.html
[prev in list] [next in list] [prev in thread] [next in thread]

Original message from the mailing list

GOALS

- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed
- Able to handle large projects like the Linux kernel efficiently (speed and data size)

COMMIT OFTEN
PERFECT LATER
PUBLISH ONCE

- Commit early, commit often
- Each commit represents one idea or one whole change (easier to read or revert)
- Each branch represents one feature (or topic) (easier to read or merge)
- Your local working directory, index and local repo are scratch pads.

UNDER THE HOOD

Git is a state of mind

WHY GOING DEEP?

- Git originally designed and created for internal-aware pro developers
  - Specific atomic commands, *NIX style
  - File-system alike level
- Untold mantra: "When you'll know what it does and how, you'll figure out how to use it"

COMMANDS: PORCELAIN vs. PLUMBING

PLUMBING(s)

Advanced, specific tools/commands for a single purpose and advanced problem solving (Barely documented: are you a Pro, aren't you?)

PORCELAIN(s)

Different grouping and use of plumbing commands (Very extended documentation)

Git: API for DVCS

Avoid duplication of API with different parameters

Split underneath logics in simpler commands (UNIX-style)

OBJECTS

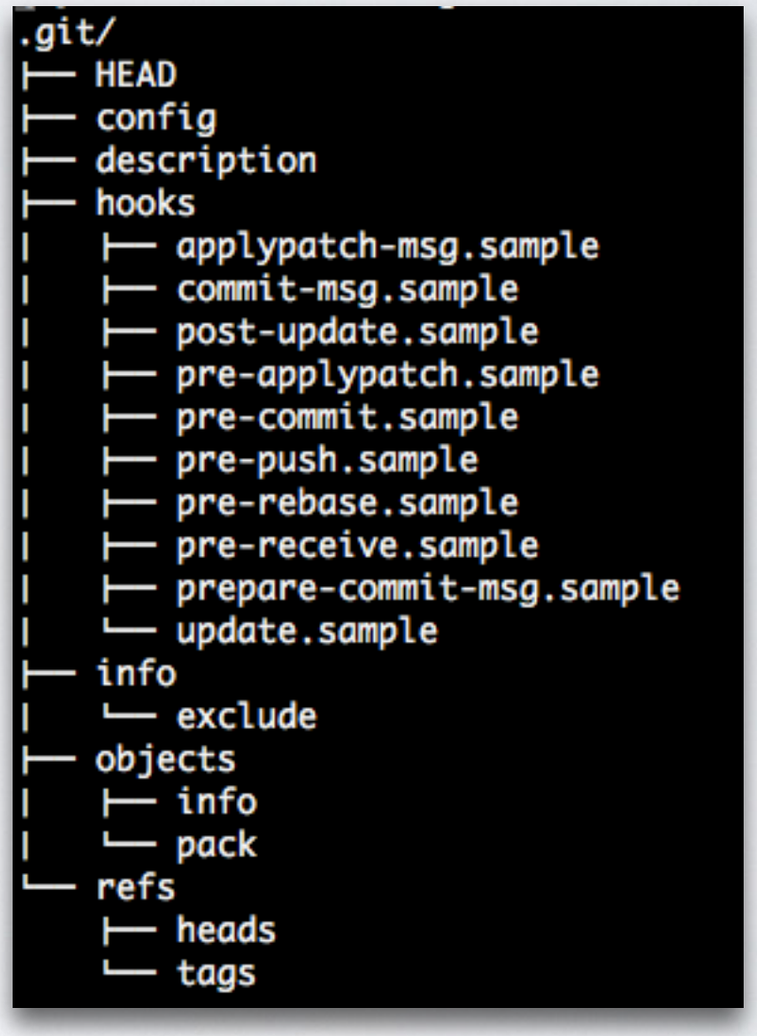
- Main element in GIT:
  - Stored in key-value database: Key => Value
  - Named after the content using SHA-1 hash: Key(Value) => Value, SHA-1(Value) => Value
    - IMMUTABLE: mutation would break hash, so references would be broken too!
    - Deduplication for free: same object referenced, not stored again
- Main object types:
  - Blob
  - Tree
  - Commit
  - Annotated Tag

OBJECTS

- Blob
  - Full content of a file without filename and metadata
  - Named after SHA-1 of content and attributes
- Tree
  - Representation of a directory structure / file(s)
  - Recursive structure: contains blob and tree references
- Commit
  - Snapshot of the repository at given time
  - Tree object reference (recursive)
  - Author's data: name, email, timestamp, timezone
  - Committer's data: name, email, timestamp, timezone
  - Commit Message
  - Parent commit reference (special commit: Merge commit has two/more than one parents)

.GIT FOLDER

- Created with ‘git init’ command
- The .git folder IS the local repository



Structure of .git folder of an empty repository

.GIT FOLDER

- After the commit of a simple text file

```
.git/
├── COMMIT_EDITMSG
├── HEAD
├── config
├── description
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   ├── pre-receive.sample
│   ├── prepare-commit-msg.sample
│   └── update.sample
├── index
├── info
│   └── exclude
├── logs
│   ├── HEAD
│   └── refs
│       └── heads
│           └── master
├── objects
│   ├── 96
│   │   └── 898574d1b88e619be24fd90bb4cd399acbc5ca
│   ├── ae
│   │   └── 7da1e4a5e4bc2ae99fdc91f0a42b5ca7b4fa31
│   ├── d7
│   │   └── bb5dbc18ca1f0a059bfda0664064915528cce9
│   ├── info
│   └── pack
└── refs
    ├── heads
    │   └── master
    └── tags
```

Structure of .git folder of the repository at the current state

HEAD

- It points to the current state of the working area
- Relative pointing to a refs, e.g.:
  - ‘ref: refs/heads/master’
- In detached state if it points to an hash
  - only good for read-only navigation of the repo, destructive otherwise

```
.git/
├── COMMIT_EDITMSG
├── HEAD
├── config
├── description
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   ├── pre-receive.sample
│   ├── prepare-commit-msg.sample
│   └── update.sample
├── index
├── info
│   └── exclude
├── logs
│   ├── HEAD
│   └── refs
│       └── heads
│           └── master
├── objects
│   ├── 96
│   │   └── 898574d1b88e619be24fd90bb4cd399acbc5ca
│   ├── ae
│   │   └── 7da1e4a5e4bc2ae99fdc91f0a42b5ca7b4fa31
│   ├── d7
│   │   └── bb5dbc18ca1f0a059bfda0664064915528cce9
│   ├── info
│   └── pack
└── refs
    ├── heads
    │   └── master
    └── tags
```

Structure of .git folder of the repository at the current state

CONFIG

- Local configurations
- Core configuration values
- Author's data:
  - Name, email, …
- Opposite to
  global .gitconfig file in home

```
.git/
├── COMMIT_EDITMSG
├── HEAD
├── config
├── description
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   ├── pre-receive.sample
│   ├── prepare-commit-msg.sample
│   └── update.sample
├── index
├── info
│   └── exclude
├── logs
│   ├── HEAD
│   └── refs
│       └── heads
│           └── master
├── objects
│   ├── 96
│   │   └── 898574d1b88e619be24fd90bb4cd399acbc5ca
│   ├── ae
│   │   └── 7da1e4a5e4bc2ae99fdc91f0a42b5ca7b4fa31
│   ├── d7
│   │   └── bb5dbc18ca1f0a059bfda0664064915528cce9
│   ├── info
│   └── pack
└── refs
    ├── heads
    │   └── master
    └── tags
```

Structure of .git folder of the repository at the current state

HOOKS

- (Sample of) scripts automatically launched in response to a specific events
- Validation, styling rules, commit checks, …
- Without the .sample

```
.git/
├── COMMIT_EDITMSG
├── HEAD
├── config
├── description
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   ├── pre-receive.sample
│   ├── prepare-commit-msg.sample
│   └── update.sample
├── index
├── info
│   └── exclude
├── logs
│   ├── HEAD
│   └── refs
│       └── heads
│           └── master
├── objects
│   ├── 96
│   │   └── 898574d1b88e619be24fd90bb4cd399acbc5ca
│   ├── ae
│   │   └── 7da1e4a5e4bc2ae99fdc91f0a42b5ca7b4fa31
│   ├── d7
│   │   └── bb5dbc18ca1f0a059bfda0664064915528cce9
│   ├── info
│   └── pack
└── refs
    ├── heads
    │   └── master
    └── tags
```

Structure of .git folder of the repository at the current state

OBJECTS

- Key-value database
- Contains blobs, trees and commits
- 256 dirs max per level with first 2 chars of object hash (avoid filesystem limits)
- pack folder: highly compressed, delta-optimized archives of objects, created at push/pull

```
.git/
├── COMMIT_EDITMSG
├── HEAD
├── config
├── description
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   ├── pre-receive.sample
│   ├── prepare-commit-msg.sample
│   └── update.sample
├── index
├── info
│   └── exclude
├── logs
│   ├── HEAD
│   └── refs
│       └── heads
│           └── master
├── objects
│   ├── 96
│   │   └── 898574d1b88e619be24fd90bb4cd399acbc5ca
│   ├── ae
│   │   └── 7da1e4a5e4bc2ae99fdc91f0a42b5ca7b4fa31
│   ├── d7
│   │   └── bb5dbc18ca1f0a059bfda0664064915528cce9
│   ├── info
│   └── pack
└── refs
    ├── heads
    │   └── master
    └── tags
```

Structure of .git folder of the repository at the current state

REFS

- Heads == Branches
- Tags (lightweight)
- Refspec: specification of references
- Heads point to a specific commit
  - Head of that branch

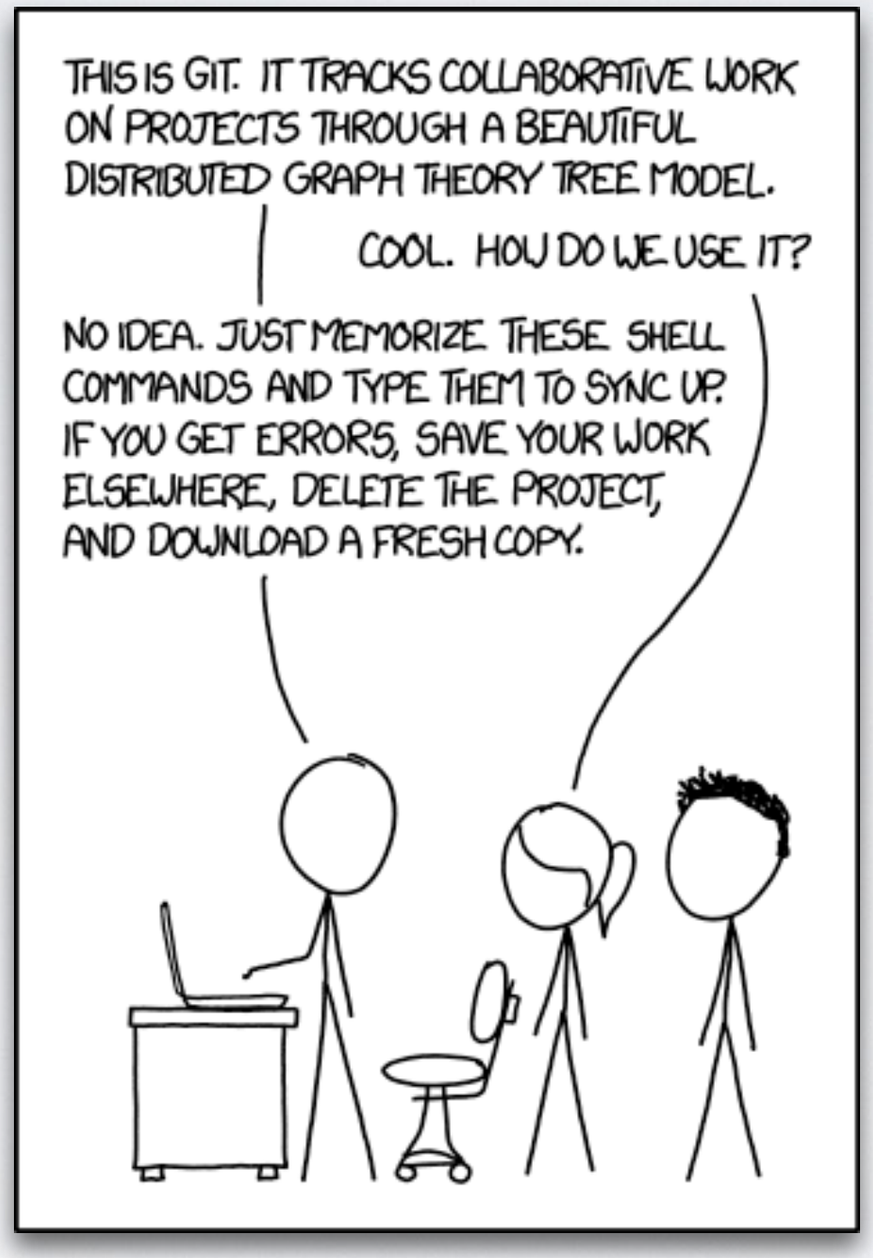
```
.git/
├── COMMIT_EDITMSG
├── HEAD
├── config
├── description
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   ├── pre-receive.sample
│   ├── prepare-commit-msg.sample
│   └── update.sample
├── index
├── info
│   └── exclude
├── logs
│   ├── HEAD
│   └── refs
│       └── heads
│           └── master
├── objects
│   ├── 96
│   │   └── 898574d1b88e619be24fd90bb4cd399acbc5ca
│   ├── ae
│   │   └── 7da1e4a5e4bc2ae99fdc91f0a42b5ca7b4fa31
│   ├── d7
│   │   └── bb5dbc18ca1f0a059bfda0664064915528cce9
│   ├── info
│   └── pack
└── refs
    ├── heads
    │   └── master
    └── tags
```

Structure of .git folder of the repository at the current state

COMMON USE

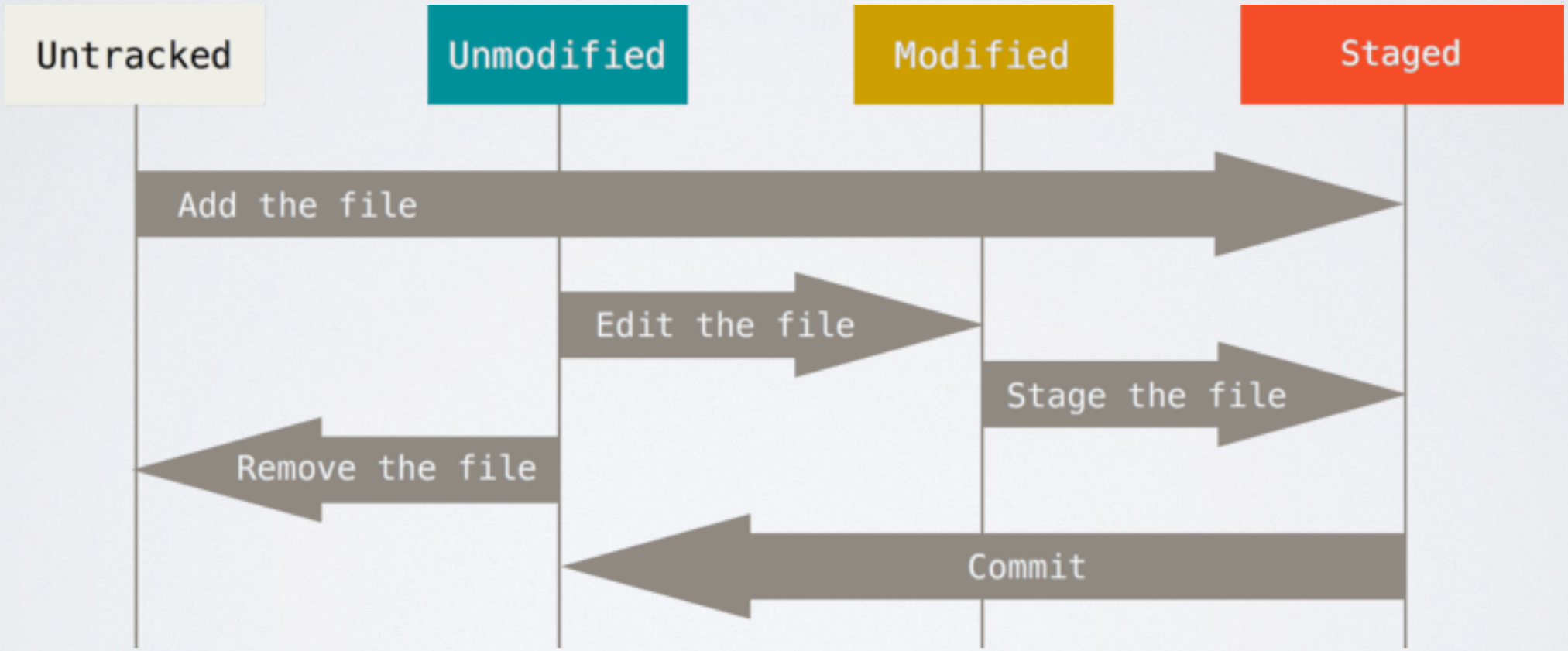
Git is a state of mind

REAL LIFE



Source and real-life story

LIFECYCLE


Untracked
Unmodified
Modified
Staged
Add the file
Edit the file
Stage the file
Remove the file
Commit

GIT GEOGRAPHY

- Local repo: .git folder and its content
- Working directory/area: brothers (and descendents) of .git folder
- Stage/Staging area/Cache/Index: (synonyms, index originally) index of changes to commit
- Stashing area/Stash: clipboard-ish area where to temporarily save the changes not to be committed, facility external to VCS

COMMON COMMANDS

- Git status
  - Status of the repository
- Git add <file, …>
  - Add file(s) to staging area
  - Under the hood: "keep an eye on it", prepare to recursively create tree and blob objects
- Git commit [-m "<commit message>"]
  - Commit the changes added to the staging area
  - Under the hood: create the commit object, associate tree, blob and commit objects
- Git pull
  - Pull the latest updates from the server
- Git push
  - Push the commit(s) to the remote server

LOG

- Git log
  - Log of all commits in the common default format
  - More or less infinite combination of parameters for different showing mode of the log (e.g. colors, graphic, relative time format: n days ago)
  - Pretty format documentation
  - One of the longest man page!
  - If lost, use A DOG: git log --all --decorate --oneline --graph

FETCH vs. PULL

FETCH

- Fetch refs (branches, tags) from remote repository (one or more)
- Doesn't download commit
- Doesn't alter the state of the working directory or local repository
- Useful for sync with new remote(s) and/or branch(es)

PULL

- Downloads commits
- For the current branch from remote counterpart
- Automatically updates the state of the working directory
- git fetch && git merge FETCH_HEAD

BRANCHES

- Extremely cheap and fast
- A branch is a file that contains the reference to a commit. That's it.
- It's not an object, so (and that's why) it's mutable!
- The entire branch is recreated recursively using parent reference of each commit, starting from the pointed one
- Fast-forward merge if the branch starts from the end of original/ destination branch (e.g. after a rebase)

BRANCHING

- Git checkout <branch>
    - Move to an existent branch
- Git checkout -b <branch>
    - Create a branch if not existent and move to it
- Git branch <branch>
    - Create a branch if not existent without moving to it
- Git merge <branch>
    - Merge the specified branch on the current one
- Git rebase <branch>
    - Rebase the current branch on the specified branch

STASHING

- Git stash [save]
  - Save the changes in the working area into the stashing area
- Git stash list
  - List all the stashes
- Git stash show
  - Show the details of a stash
- Git stash apply
  - Apply the specified stash on the current working directory
- [other options and combination…]

TAGS

- Arbitrary fixed reference to a commit
- 2 types (completely unrelated to each other)

Lightweight

- Structurally similar to branch
- Simple name pointing to a commit

Annotated

- Structurally is an object
- Simple name pointing to a commit, with a message

TAGGING

- Git tag
  - List all the tags
- Git tag <tag name>
  - Create a lightweight tag on the current commit
- Git tag -a <tag name> -m <message>
  - Create an annotated tag on the current commit, with the specified message
- Git show <tag name>
  - Show details about the specified tag
- Git push [remote] --tags
  - Push the tags on remote repository (Not pushed by default push command!)

BLAME

- Useful for firing people
- Shows the changes made on an object (list of commits) with relative authors
- Syntax:
  - git blame [lots of options] <file>

ADVANCED USE

Git is a state of mind

REFLOG

- Reference logs :)
- git reflog [options]
- “Record when the tips of branches and other references were updated in the local repository.” (source)
- “It saves your… life!” (anyone who’ve used it)
  - Commits are not deleted until the garbage collector comes in action!
  - Go back in time, locate the commit and restore

CHERRY-PICK

- Apply the changes introduced by an existing commit (somewhere in the repo)
- It creates a new commit
- Best results if the commit is atomic and self-explanatory
- git cherry-pick <commit>

REBASE

- Reapply commits on top of another base tip
- Interactive rebase with -i option (options in text editor)
- Main uses:
  - History rewriting, actions (edit, squash, delete, …) on the same base: git rebase <commit>
  - Branch update, git rebase <branch>:
    - B branched from A at commit C1, A updated after the divergence (commits C2, C3)
    - Rebase B on top of new tip of A (C3) reapplying all the commits of B on top of C3
- Best results if the commit is atomic and self-explanatory

BISECT

- Use binary search to find the commit that introduced a bug
- Automatic mode usage:
    - Select known good commit: git bisect good <commit's sha-1>
    - Select known bad commit: git bisect bad <commit's sha-1>
    - Run the test in commits between "good" and "bad": git bisect run <bash script>
    - Bisect will stop at the first commit that introduced the bug (test fails)
- More info

MODULARITY

Git is a state of mind

SUBMODULE

- Include a git repository inside another one
- Both stories remain separated
    - Push/pull, branches to/from original separate repository
- .gitmodule
    - Reference to submodule(s)'s repository
    - Reference to submodule(s)'s current commit (HEAD)
    - Reference to submodule(s)'s destination directory (added empty to parent repository)

SUBMODULE COMMANDS

- Git submodule add <repository> [path]
    - Add an existing repository as submodule in path or subrepo name if any
- Git submodule foreach '<bash>'
    - Execute the command specified for each submodule added in the parent repository
- Git submodule update [path]
    - Update the submodule to the state specified into .gitmodule file (potentially discard new changes)
- (Canonical add, commit to update the .gitmodule file with newest submodule commits)

SUBTREE

- Include a git repository inside another one
- Stories are not separated: the subtree's story is included in parent repository (can be squashed)
- Modification to subtree's code goes on parent project's repository
- Can pull updates from original repository
- Can push updates to original repository

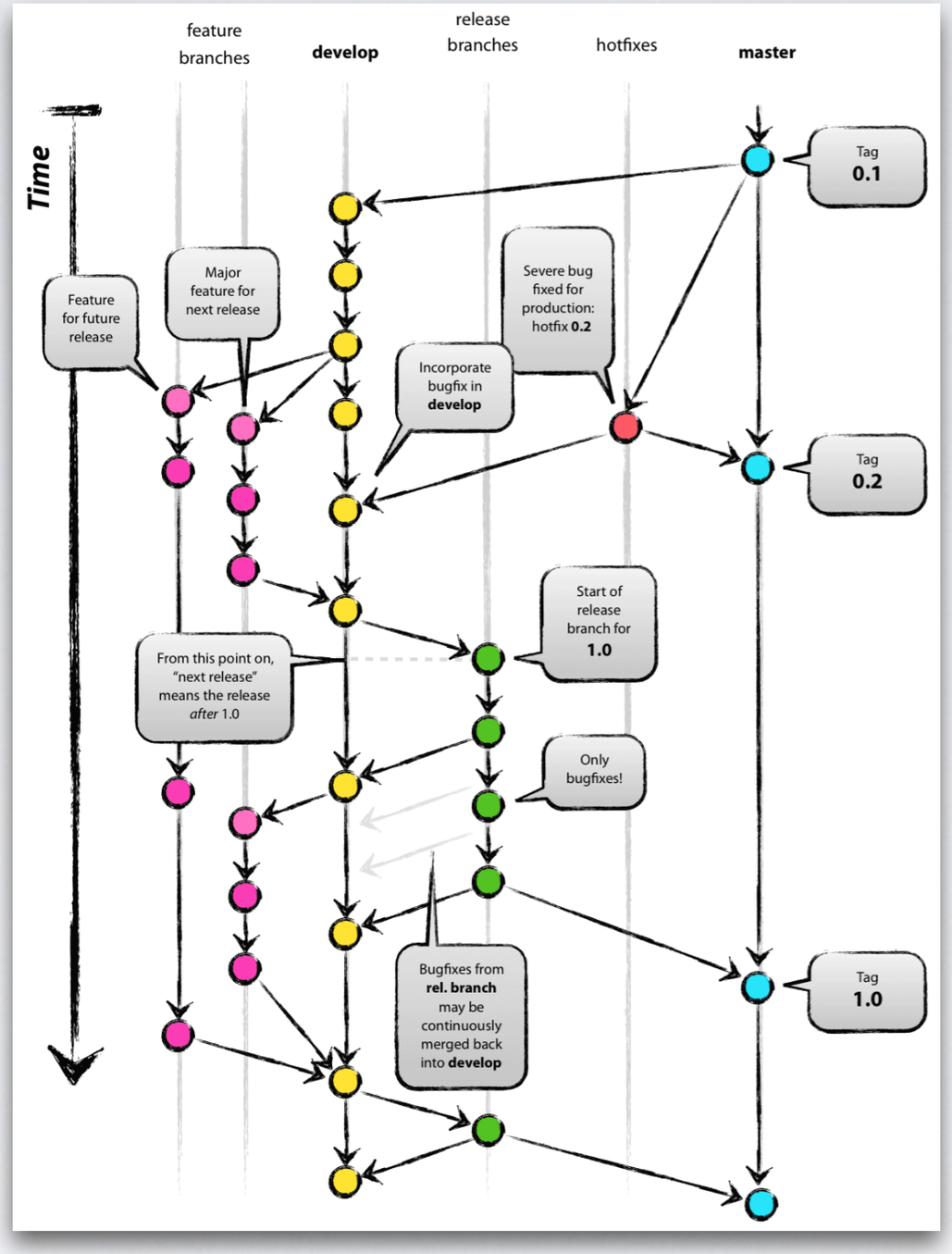
SUBTREE COMMANDS

- Git subtree add --prefix=[path] <repository> <branch> --squash
  - Add an existing repository as subtree in path squashing all the commits in one
- Git subtree pull --prefix [path] <repository> <branch> --squash
  - Update the subtree with new commits on remote repository
- Contribute back to remote repository:
  - Add the project as another remote: git remote add <name> <repo>
  - git subtree push —prefix=<prefix> <remote> <branch>
- More info about subtree: Source, Source

WORKFLOWS

Git is a state of mind

GITFLOW



Source, from GitHub

ECOSYSTEM

Git is a state of mind

BITBUCKET

More than 1 million teams and 6 million developers, Git/Mercurial

Homepage

GITHUB

Homepage

GITHUB

- Biggest community exclusively for git ecosystem
- 316 programming languages, ~6M active users, ~332K active organizations, ~19.5M active repositories
- Main home for any open source project (hosted for free with TravisCI)
- Most advanced web interface and support for most diffused and evolved development metodologies at any level (Agile, Testing, CI/CD, …)
- Highly reliable, but not self-hostable
- The annual state of the Octoverse

GITLAB

GitLab

Homepage

GITLAB

- Fastest development project and community exclusively for git ecosystem: rolling releases and CI
- More than 100K active organizations self-hosting (2/3 market), ~80% of mobile developers repositories
- Self-hostable (Vagrant, Docker, native), free public and private repositories with unlimited members
- Awesome repository management (user permissions, branch protection, Webhooks, …)
- Most advanced web interface and support for most diffused and evolved development metodologies at any level (Agile, Testing, CI/CD, …)
- Most experimental and cutting-edge solutions for Agile and DevOps metodologies development (Auto DevOps, …)
- Full stack of additional features (pages, CI/CD, Cycle, Issue Tracker and Board, Review, …)

EXTRAS

Git is a state of mind

EXTRA

- "GitHub cheatsheet" - Official basic GitHub cheatsheet
- "Visual cheatsheet" - Interactive cheatsheet
- "Ungit" - Interactive web interface
- "Gource" - Repo visualization
- "GitKraken" - GUI tool
- "GitHub Desktop" - Official GitHub GUI tool

REFERENCES, RESOURCES, LICENSE

Git is a state of mind

REFERENCES / RESOURCES

Conferences

"Git From the Bits Up" - Tim Berglund

"Advanced GIT for Developers" - Lorna Jane Mitchell

"Get Started with Git" - David Baumgold

"Advanced Git" - David Baumgold

"A journey into" GIT internals with Python - Andrey Syschikov

Books

Pro Git

"Git Best Practices Guide" - Eric Pidoux

"Mastering Git" - Jakub Narębski

"Learn enough Git to be dangerous" - Michael Hartl

LICENSE

- The whole presentation and the entire content (except where alternatively and explicitly specified) is property of the author, Nicola Costantino, and it's released under the term of the "Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License"