Introduction to Revision Control

The palest ink is better than the best memory

Chinese proverb

Bartosz Kostrzewa

GK1504 block course, 1st October 2014, HU Berlin

Learning Outcomes

Lecture

- Understand **what?** and **why?** of revision control
- Appreciate different **systems** and **workflows**
- Learn where to get **help**
- Choose how it can be useful for **you**

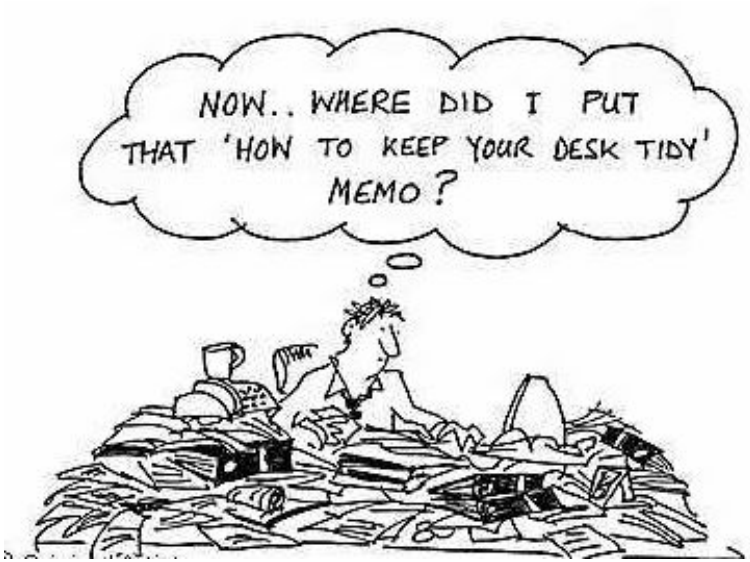
Tutorials

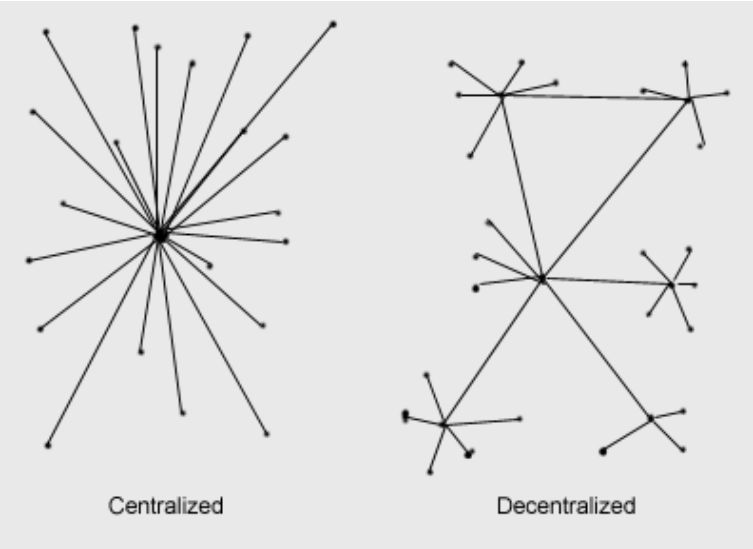
- Intro to **git**
- Basic steps
- More advanced: learn by example

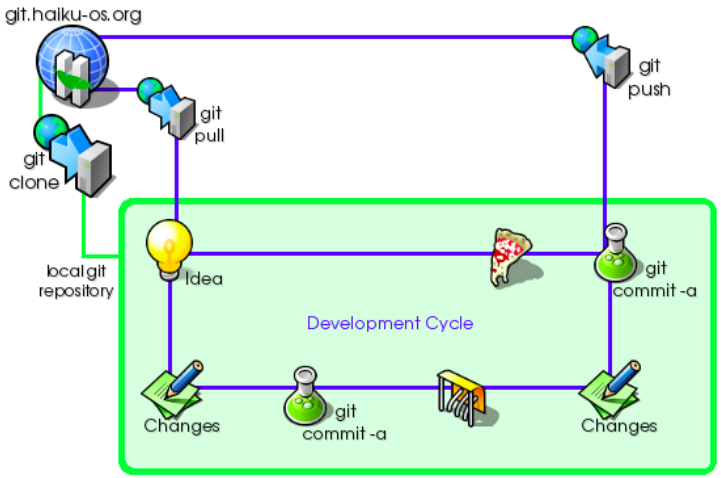
Questions

Will try to pause frequently, if urgent, please interrupt me!
Tell me to slow down if too fast!

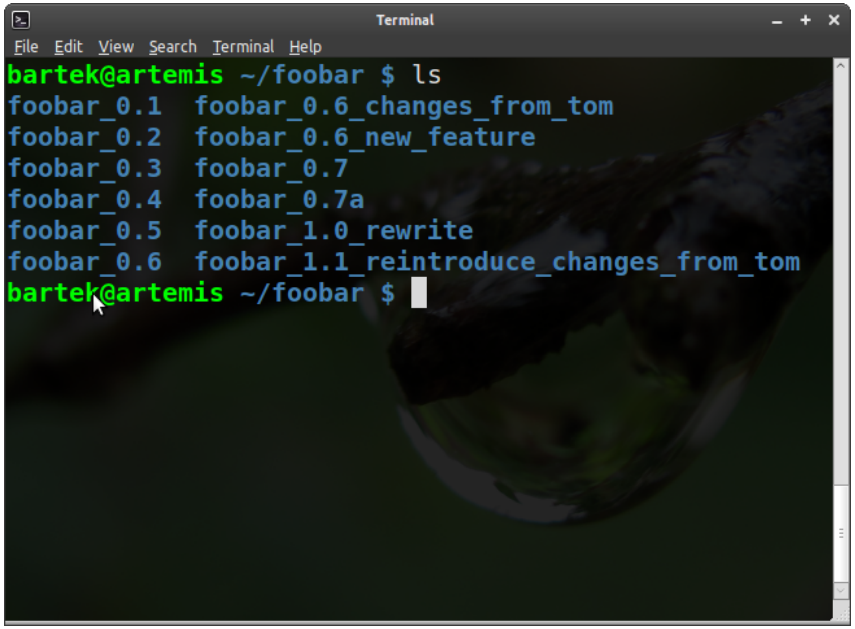
Outline


NOW.. WHERE DID I PUT THAT 'HOW TO KEEP YOUR DESK TIDY' MEMO?


Centralized
Decentralized

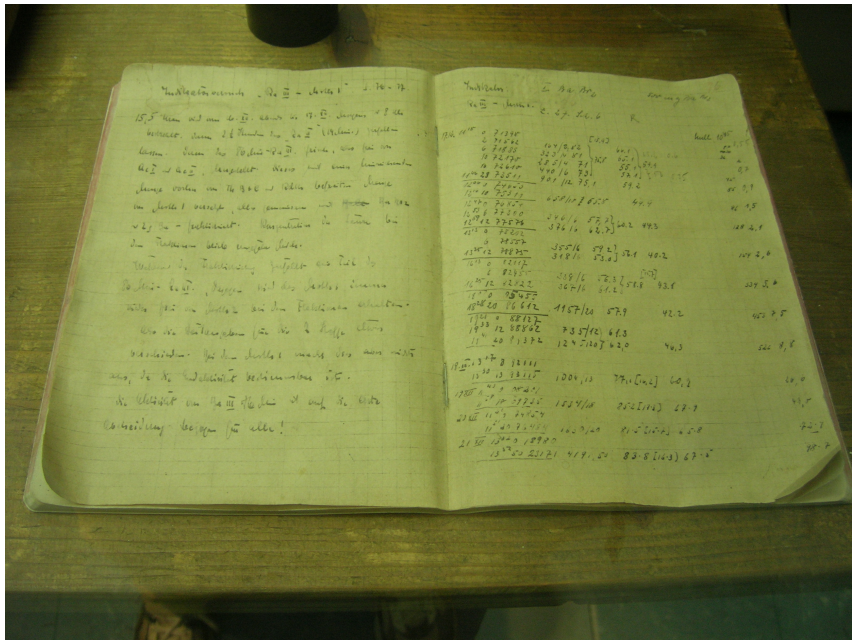

git.haiku-os.org
git push
git pull
git clone
local git repository
Idea
git commit -a
Development Cycle
Changes
git commit -a
Changes

Revision Control



- Log changes
- Document development progress
- Understand impact of bugs
- Team management
  - shared development
  - code review
  - access restrictions
- Release management
  - also: management of bugfixes for multiple versions

Revision Control in Science



wikipedia.org/wiki/Lab_notebook

- In science revision control must have a special significance!
- You can think of it as a "lab notebook" for programs and papers.
  - Organize research output and progress
  - Make research reproducible
  - Document "experimental apparatus" and methods

What is a Revision Control System? (RCS)

Version Control System (VCS) or Source Code Management [System] (SCM[S])

Basic Features

- Store source-code and make it possible to move around in history
- Keep meta-information
  - When was a file created/modified?
  - By whom? What was changed?

Advanced Features

- Follow changing directory structures
- Record merges, splits and replacements
- Ease deployment of new features
- Help with structured programming workflow
- Assist in debugging
- Release management
- Help with collaborative development

Revision Control Features

Branches

basic branching

You can think of branches like "virtual directories"

- copies of source code, branched of at some point in time
- maintain multiple versions of code in parallel

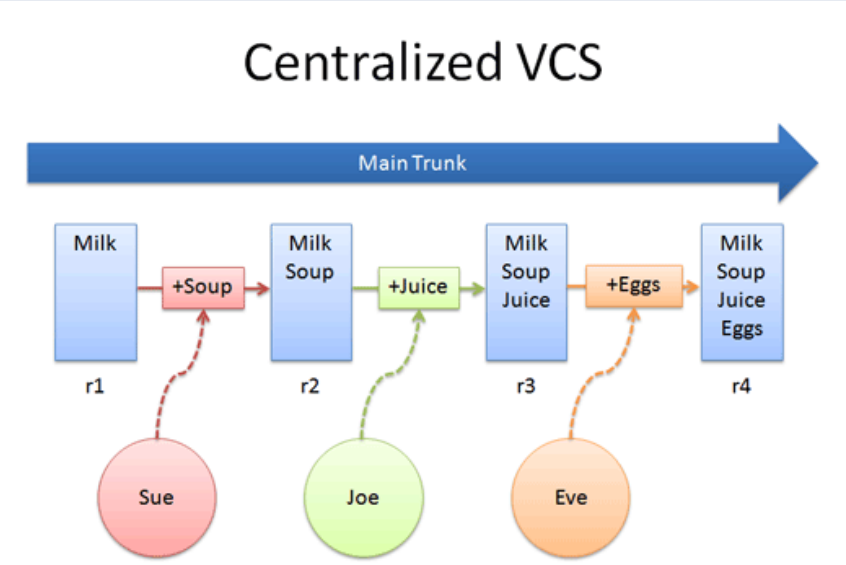
advanced branching

- branch off to test an idea
- write a bug fix
- merge features from different branches

Centralized and Distributed Revision Control Systems

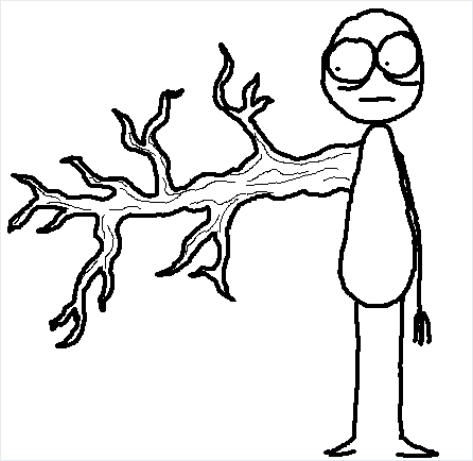
"The Cathedral and the Bazaar"

CVS, SVN, ... - Centralized Revision Control



- Repository on central server
- Update, work, commit, [resolve conflict...] workflow
- Revision history often stored as complicated Δ-information

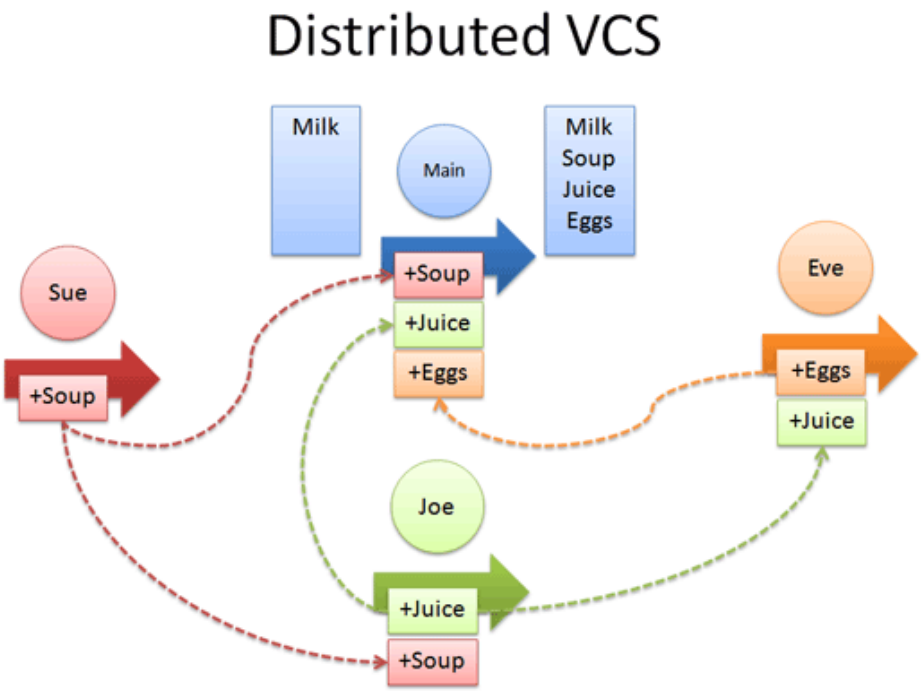
- Branching difficult (expensive)
- Merging difficult or impossible
- What if server down?



These systems are great, but are they really well-suited for our work?

Centralized and Distributed Revision Control Systems

"The Cathedral and the Bazaar"



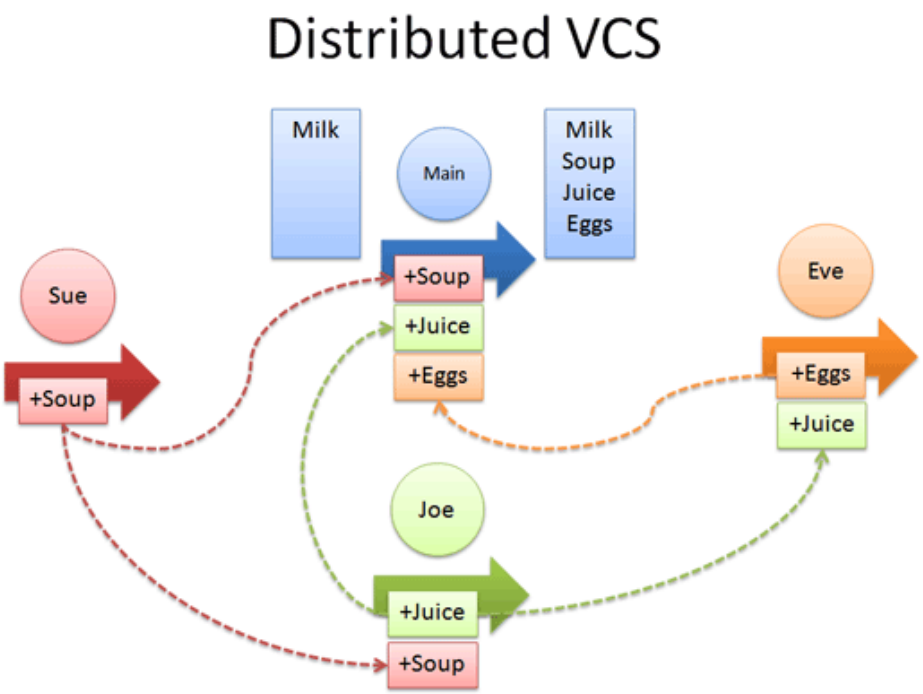
git, bzr, hg, ... - Distributed Revision Control

- Local
- Independent
- Flexible
- Allow many different workflows
- Well-suited for teams that rarely meet in person and work at (possibly) widely different times

- Inheretly more complicated
- Can get confusing

Distributed Revision Control Systems

Equivalence of Repositories



Clones!

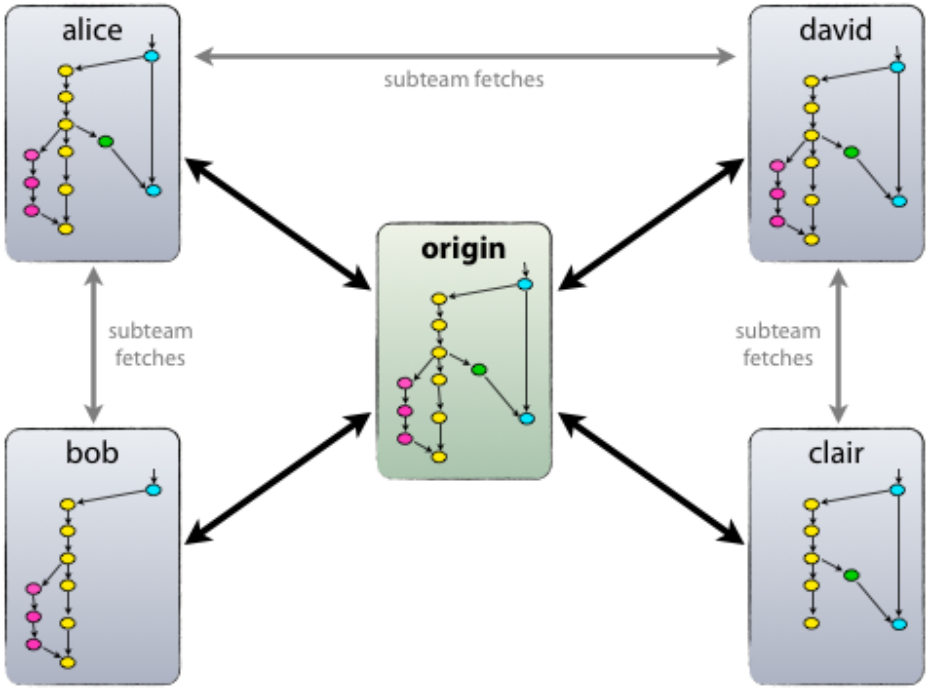
- Everyone can pull from everyone else
- Every repository is a backup!
- Change tracking is local → fast

More Traditional

- You can designate one repo to be "central"
- Clones are sometimes called "Forks"

Distributed Revision Control Systems

With a Central Repository

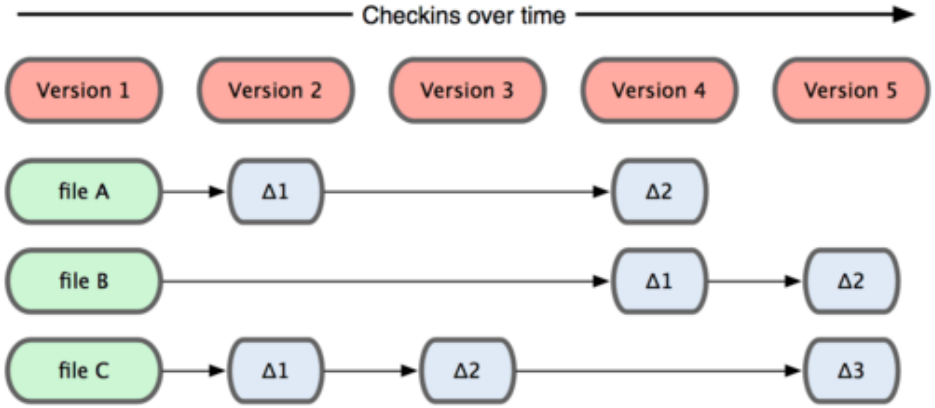


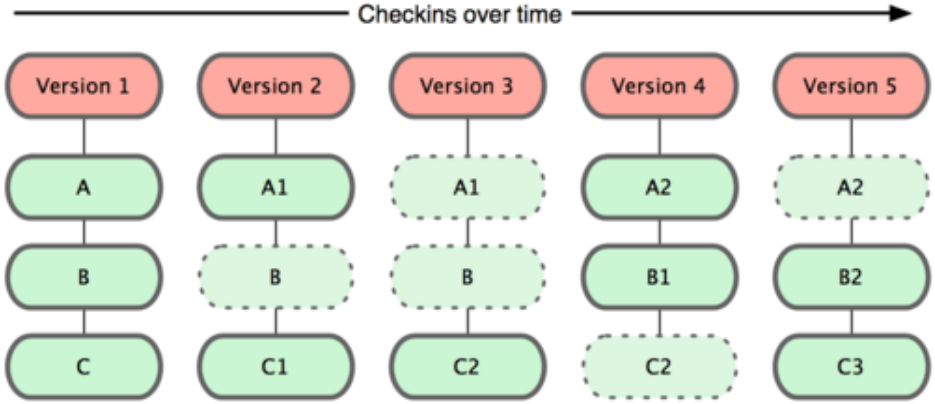
Most Common Model

- All repositories are still equivalent
- Release Management can now be done on "origin" repo

git - the stupid content tracker

storage paradigms - differences vs. snapshots





Δ storage

- store incremental differences
- seems like a good idea, but:
  - merging can be a pain
  - what if you refactor (split a file into many)
  - can get very slow...

snapshot storage

- store snapshots of entire source code at each point in time
- moving btw. snapshots is fast!

Δ storage makes refactoring, branching and merging potentially difficult!

git - the stupid content tracker

revisions vs. commits

SVN - Revisions

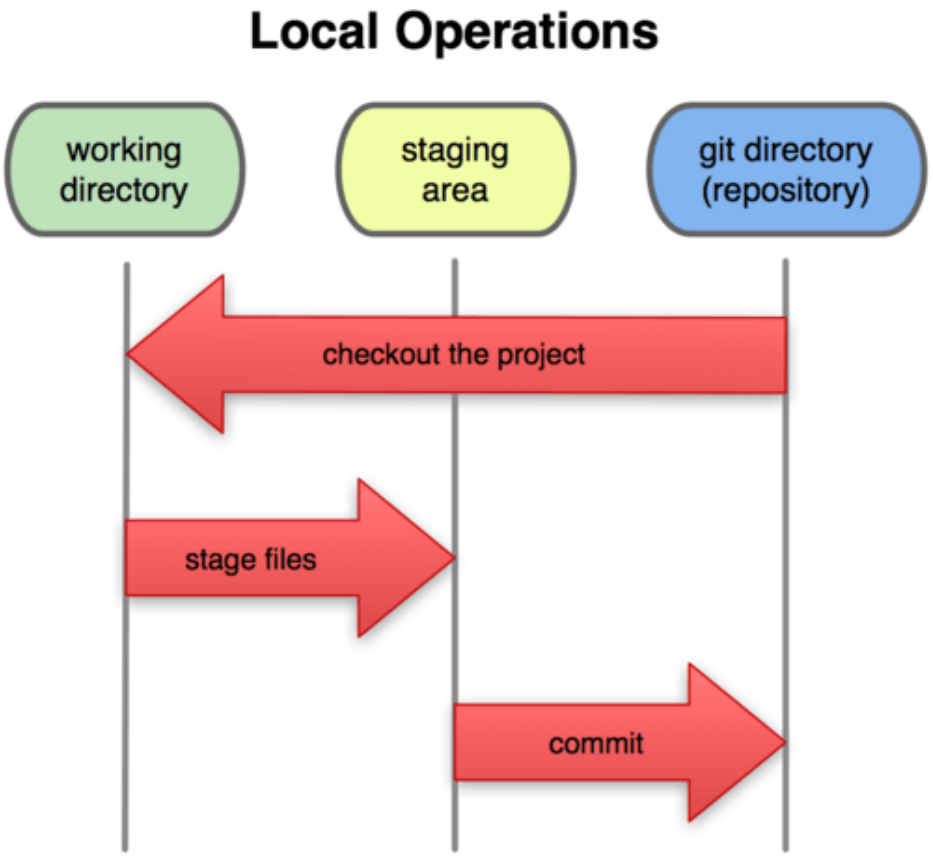
- Mark point in Δ-information time - like a version number
- since there is one repository, easy to remember
- relatively expensive $\rightarrow$ encourages very large checkins with hundreds of changes

git - commits and unique IDs

- every object (commit, file, ...) has a unique ID: 532e9cc13a593e7f8ab016107f6945e33967b981
- a commit identifies the changes made between two "snapshots"
  - can be extracted and applied to different repository or branch
- really cheap - you can develop new features with small easy to understand commits
- hard to remember, but partial matching: 532e9 should be enough

git - the stupid content tracker

the staging area



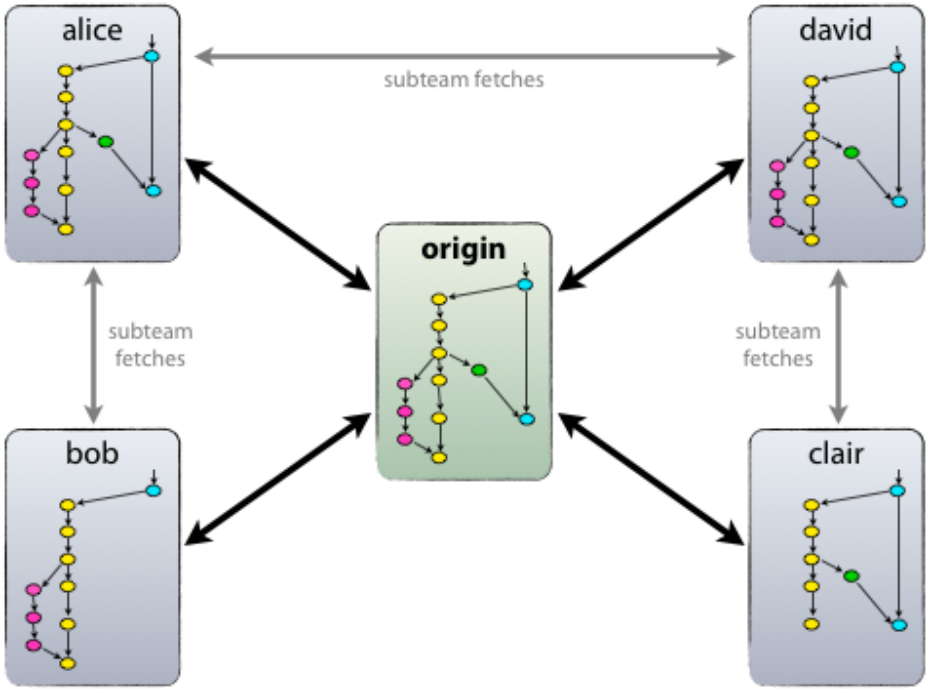
genesis of a git commit

git offers a so-called staging area for constructing commits

- make changes to code (possibly many)
- group changes logically (`git add`)
- commit change groups (`git commit`)
- often spouts of creativity, may change many files
  - Allows organized, small, understandable commit *à posteriori*

git - the stupid content tracker

remote repositories



Anything can be a Remote

- Different directory on same machine
  - share repositories via e-mail, cloud storage
- share repositories via network file systems
- git server = machine you have ssh access to

Public (read-only) git

- public directory on web server
- special git server

git - the stupid content tracker

specialty: cheap branching

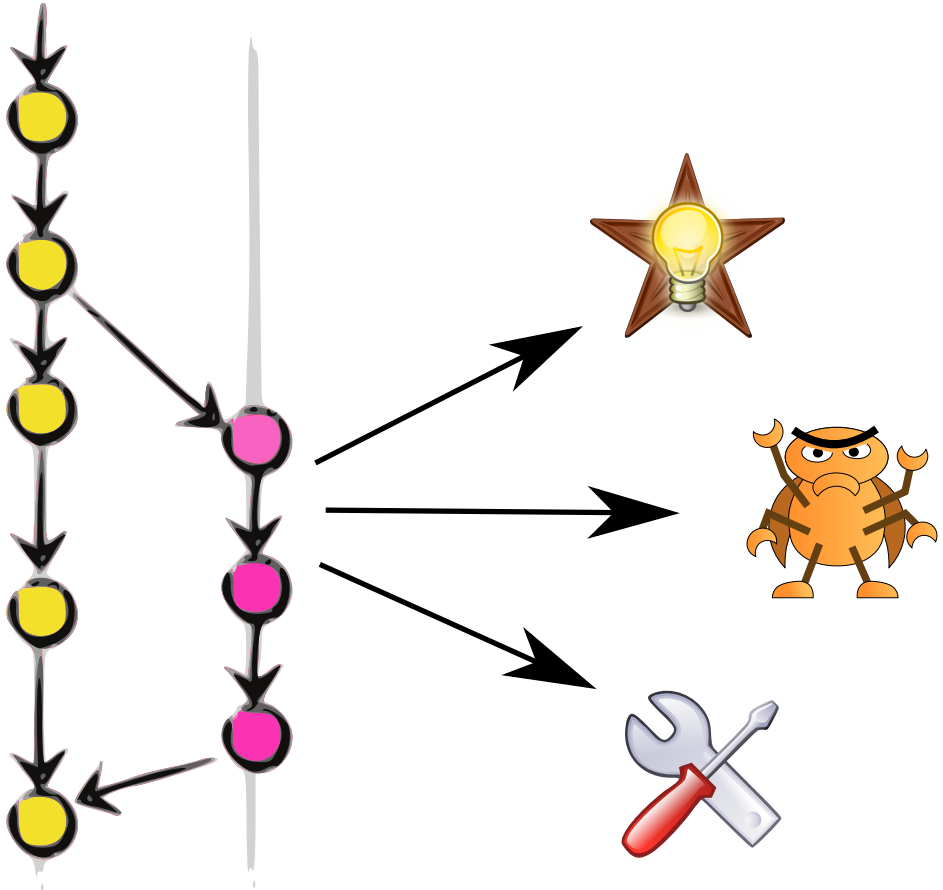
branches are ubiquitous!

git offers extremely cheap branching and merging

- new ideas
- bug fixes
- maintenance

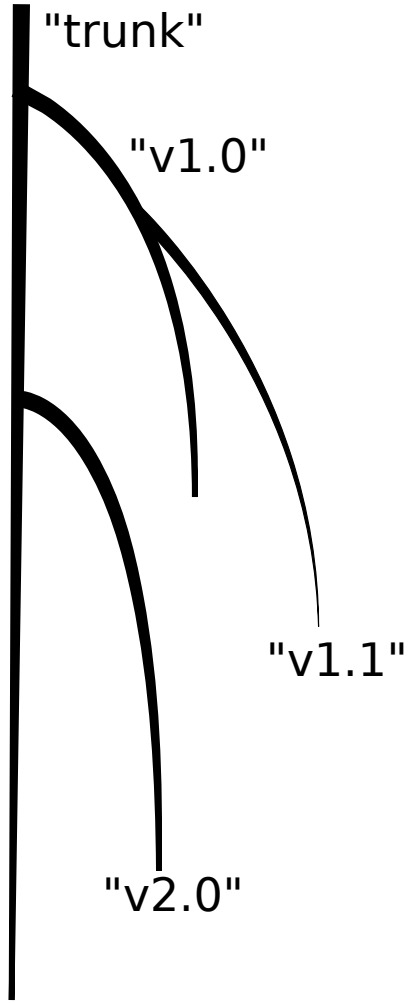
branches are useful!

- commits from one branch can be applied to another
- aid with clean, logical programming
- prevent junk from accumulating in main branch



Branching Development Models

Trunk, Release



Often used for commercial software

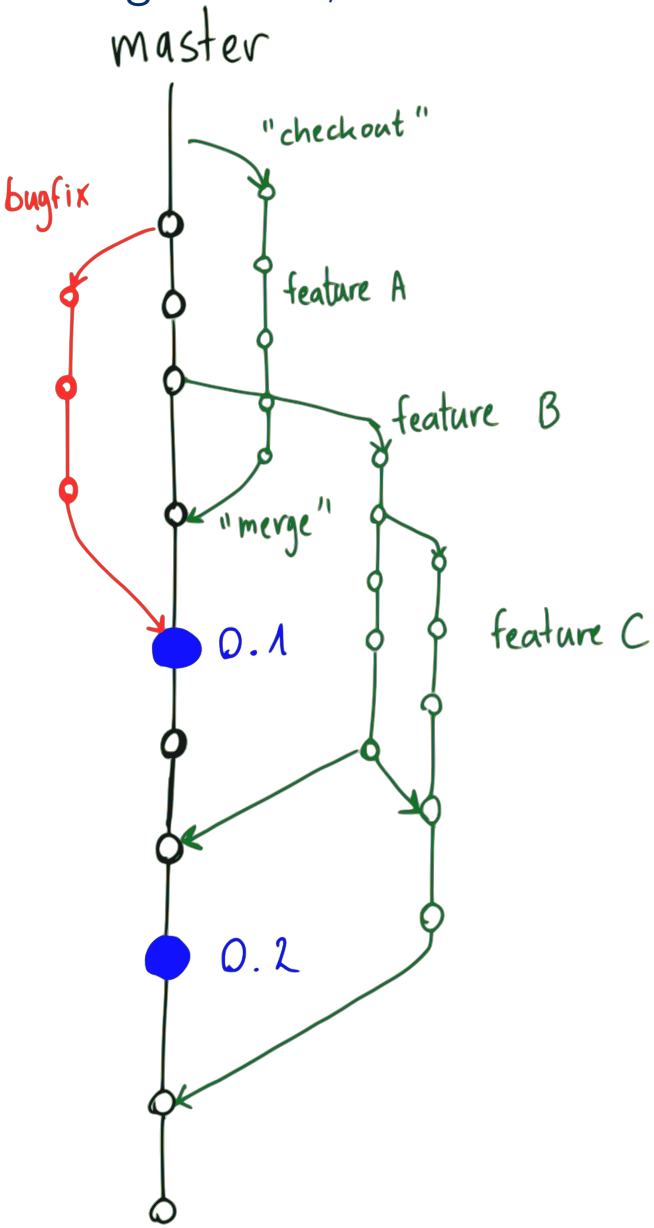
- development takes place in trunk, everybody has write access
- may alternate between stable and unstable

Benefits & Problems

- centralized - easy to see what is "current"
- works well in teams that meet often and can avoid conflicts
- without meetings, conflicts may be frequent and hard to manage
- what if devs have vastly different experience levels? $\rightarrow$ code review becomes problematic

Branching Development Models

Rolling Release, Feature Branches



Approach often chosen for open-source software

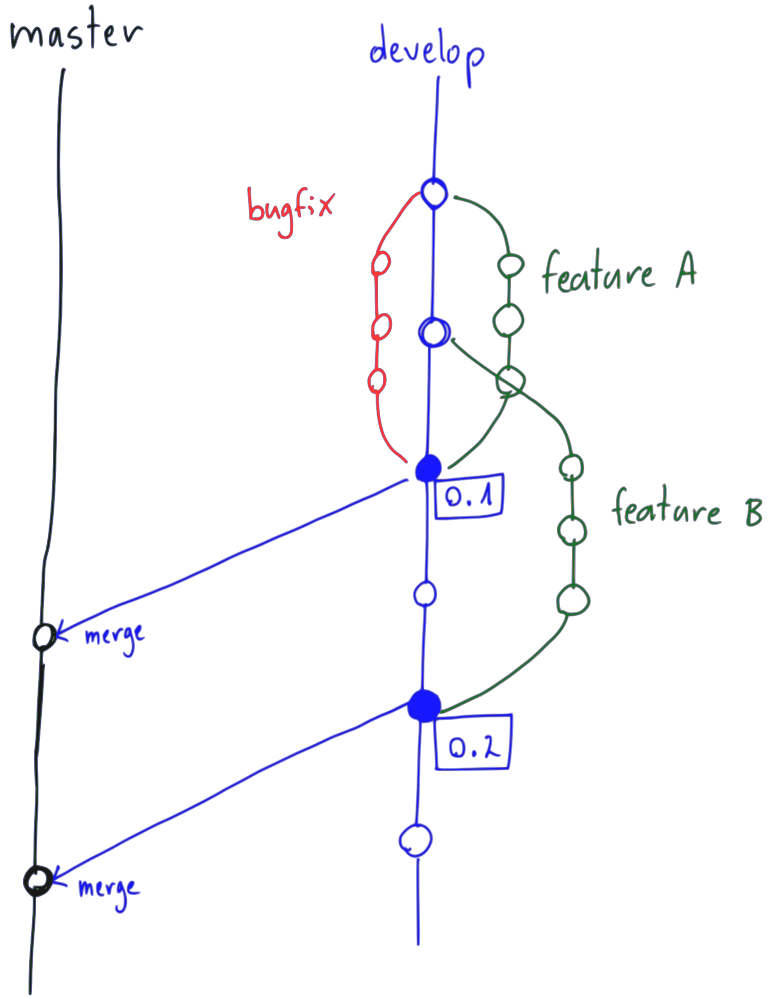
- New features and bug-fixes developed in branches
- Stable releases marked with "tags"
- Goal: keep code as runnable as possible at all times

Benefits & Problems

- Well-suited for distributed teams
- Simplifies code review
- Enables experimentation
- Can lead to "stale" branches
- Ill-suited for monolithic code

Branching Development Models

Stable Branch, Development Branch, Feature Branches



Combines trunk/release model with feature branches

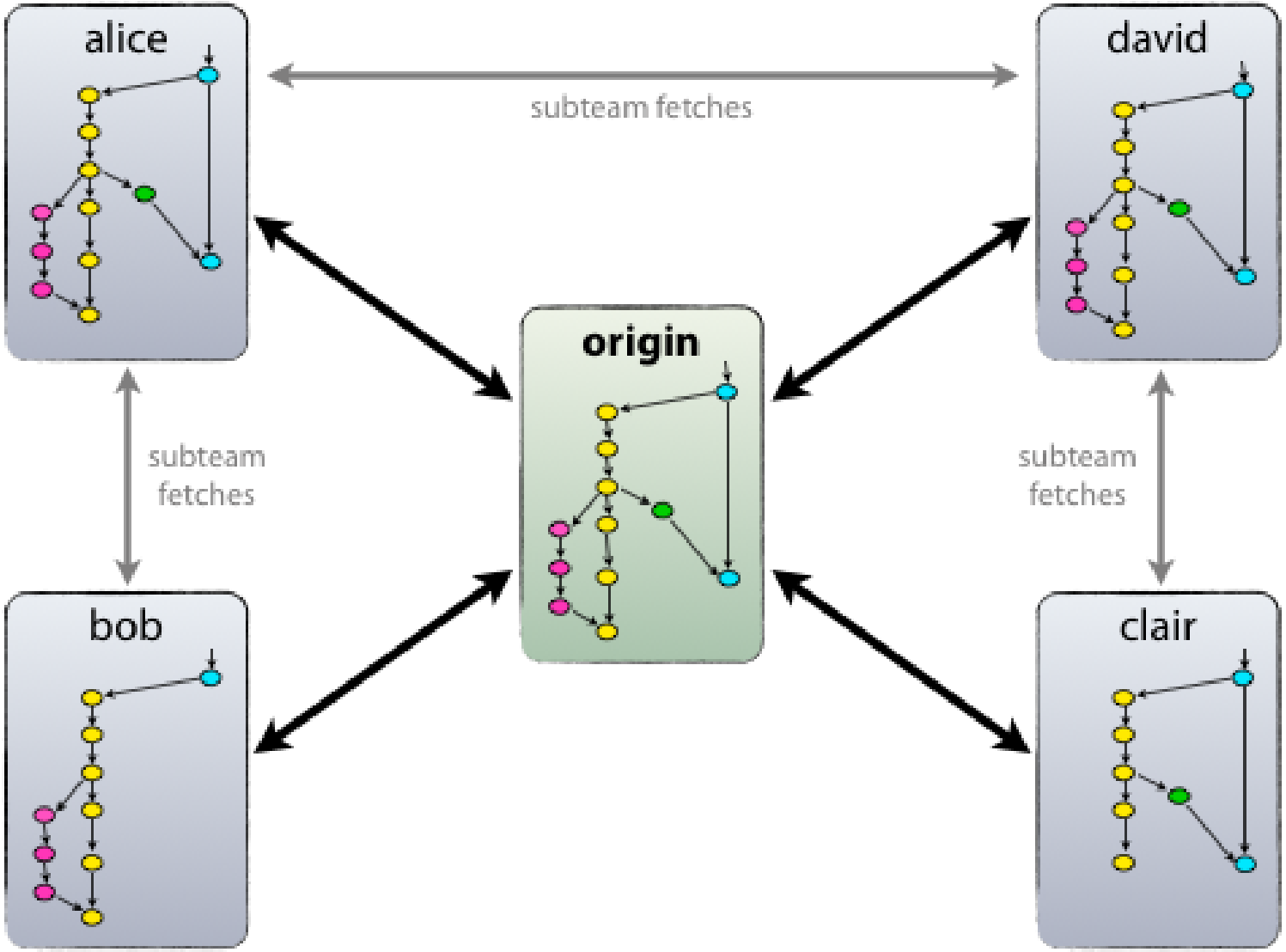
- "master" and "develop" branches
- "master" always current stable release
- "develop" not necessarily in runnable state

Benefits & Problems

- Easy to see what is "current"
- Shares benefits and drawbacks with both previous models
- May encourage sloppiness

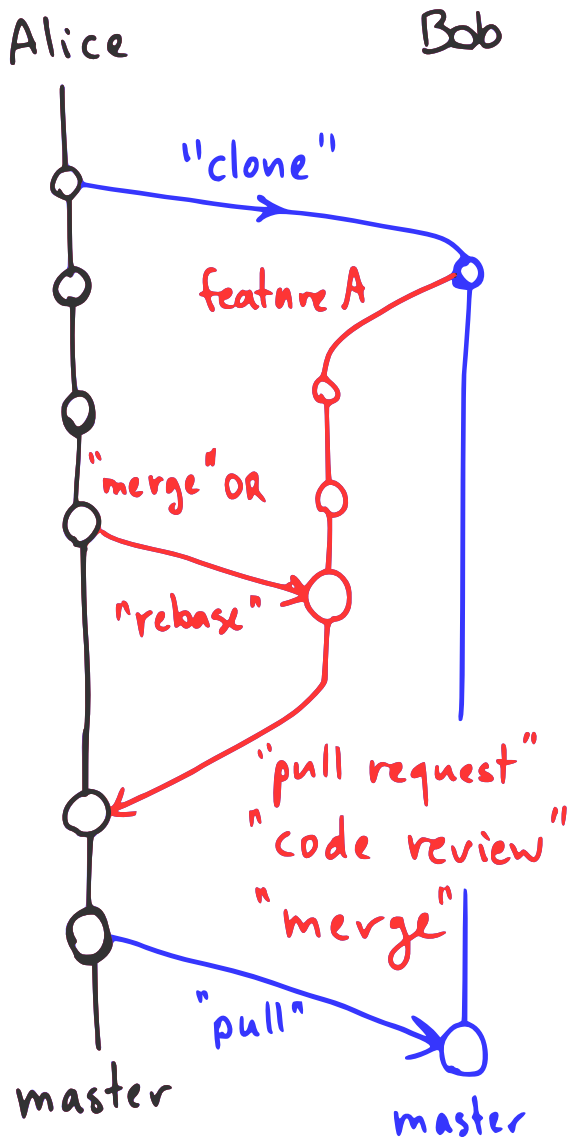
Remotes and Branching Models

Revisiting the "Central" Repo



Remotes and Branching Models

What is a "pull request"?



Alice is the team leader for project X and Bob has implemented a new feature.

Bob: Hey Alice, finally got around to implementing feature A.

Alice: Great, I took a look, could you please add some comments to routine F?

Bob: Okay, will do, I'll also *merge* with your current *master* so you can *pull* it in.

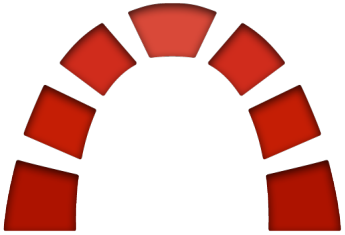
Alice: Much better, next time maybe consider *rebasing* rather than *merging*.

- code review part of development process
- needs a little more work from Alice
- this workflow is optional, of course

Source Code Management and Collaboration

redmine & github

Interface combining source code hosting, project management and issue tracking desirable → link bug reports and development work



redmine

Self-hosted

- open-source
- well-intergrated, lots of plugins
- compatible with various RCS
- administrative burden to keep up-to-date

github

Online/Commercial

- free for open-source software
- all repositories fully public
- hassle-free but what if provider goes bust?
- private hosting for a fee

Summary

Tried to convey:

- What is an RCS?
- Why would I want to use one?
- What is git and why is it special?
- How do you work with an RCS?

Exercises

This was all quite abstract, exercises will cover everything again with concrete examples.

$\Rightarrow$ you should identify how RCS can be useful for you

Thanks for listening!

See last slide for some more information.

More information

Where to get help?

- SVN users: `http://git-scm.com/course/svn.html`
- `http://git-scm.com/about` has a really nice quick outline of git features
- git pro book: `http://git-scm.com/book`
  - free to read online
  - first three chapters cover all the basics
- stackoverflow Q&A: `http://www.stackoverflow.com` has a lot of solutions for common git use-cases

git on Windows

- if you're a Windows user, `http://msysgit.github.io/` offers a very capable Windows native git implementation with bash emulation and graphical interfaces
  - `http://guides.beanstalkapp.com/version-control/git-on-windows.html` provides an installation and usage guide