





Version Control and Configuration Management

Peter C Rigby


Configuration Management

- Continually changing product 
- Released product is different from source
- Must be able to recreate old versions of the system
 - e.g., fix a bug in a release that a client is using
 - backporting
- Trace the history of a file, function, etc
 - Design/program comprehension

Terminology

- Commit: To store a change in the version control system in such a way that it can be incorporated into future releases of the project 
- Log Message: A detailed description describing the change, can include reviewer information and links to bugs and other artifacts
- Repository: A software system that stores the change history
- Push/Commit: Send commit to another repository
- Update/Pull/Checkout: Obtaining a copy of the project from a repository


Terminology (cont.)

- Head: The most recent commit to the repo
- Changeset/Commit: A logically independent change
- Patchset: A set of commits 
- Branch/stream/codeline/tree: A line of development that is isolated, so that changes made to the lines don't affect it and vice versa


Some suggestions

- Version everything (source code, web pages, documentation, FAQ, design notes, etc...)
- Any piece of information worth writing down is worth versioning
- Things that don't change should be archived
 - If you use git, you can ignore this recommendation
- Don't keep generated files under version control
- Use branches to isolate and group changes

Isolation

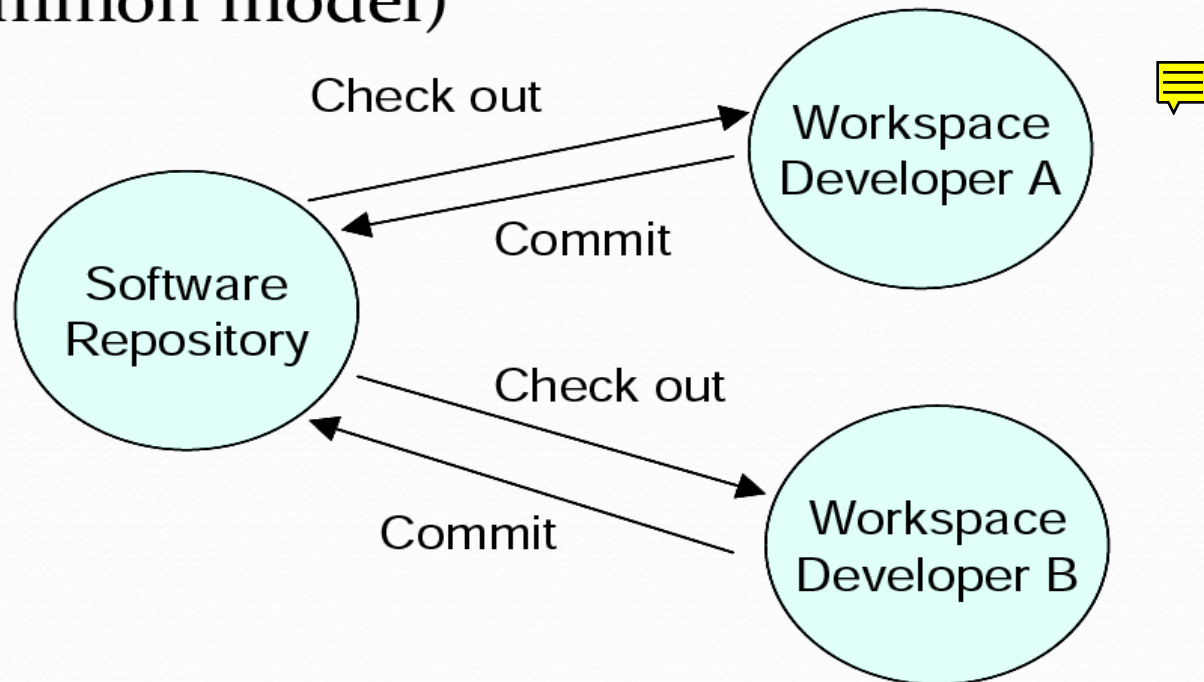
- Good isolation
 - Developers work in parallel without making conflicting changes
- Bad isolation 
 - Developers unknowingly make conflicting changes
 - Obvious conflicts (e.g., change the same line)
 - Violations of architecture or API (more serious bugs)

Types of version control

- Centralized Version Control (CVC)
 - CVS, SVN, Perforce
 - Central store of history, devs only have particular versions 
- Distributed Version Control (DVC)
 - Hg, Bazaar, git
 - Can also function as a CVC
 - No inherent differences in repositories, every dev has full history of the system


Central Repository

- Central repository / distributed workspace (most common model)



Optimistic strategy (multiple check-outs – possible conflicts) or Pessimistic strategy (locks, no conflicts)

“Work of the head”


- The most recent commit to a VCS is called the head
- Advantages
 - Working with the latest up-to-date code
 - Avoids major merging and integration conflicts
 - e.g., Apache, subversion, CVC
- Disadvantages
 - **mixes integration and development changes** and causes developers to be constantly distracted by unrelated changes (i.e., insufficient isolation)
 - Code is less well tested, preliminary and often unstable and unfamiliar
 - Debugging/testing: did I break it or did someone else!?

Work off of Stable Tags or Releases



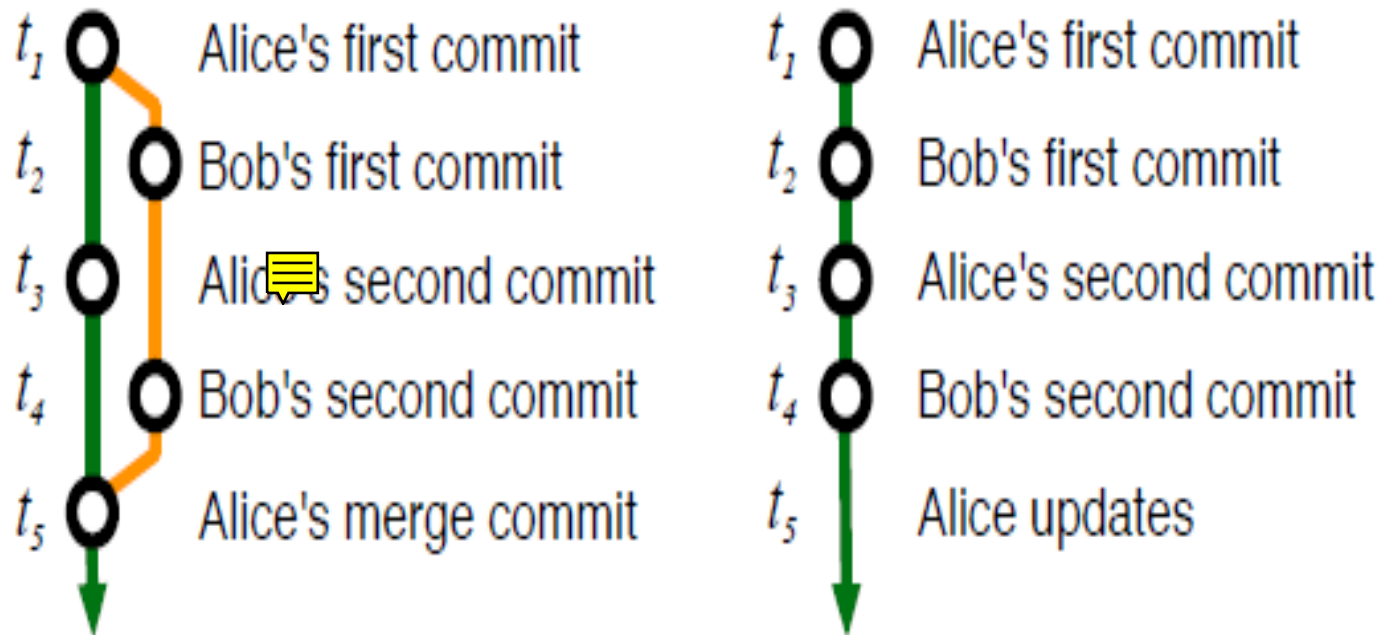
“You should never pull my tree at **random points [i.e., head]**. It makes your tree just a random mess of random development. . . . You also **lose a lot of testability** since now all your tests are going to be about all my random code. . . . **sync up with major releases**, ... [a] non random point.”, Linus Torvalds

“Work of Named Stable Bases”

- Bases, tags, releases are of a known stability, set features and level of testing (no randomness!)
- Advantages 
 - Developers are isolated from unrelated changes while they work
 - If something breaks, you broke it
 - Deal with familiar code (sandboxes)
- Disadvantage
 - The longer you wait the more likely you are to end up in “merge hell”, so “merge early, merge often”
 - Merges are usually done more frequently between subgroups

Branches vs Head

“Working off head” effectively requires a merge on every commit




Why doesn't everyone use branches?

Merging is too “painful” and “messy”

“We had branches for versions [releases]. Feature branches were VERY rare for us.”,
Koziarski (Rails)

Branches are hard in traditional version control systems, so they were rarely used. Why?

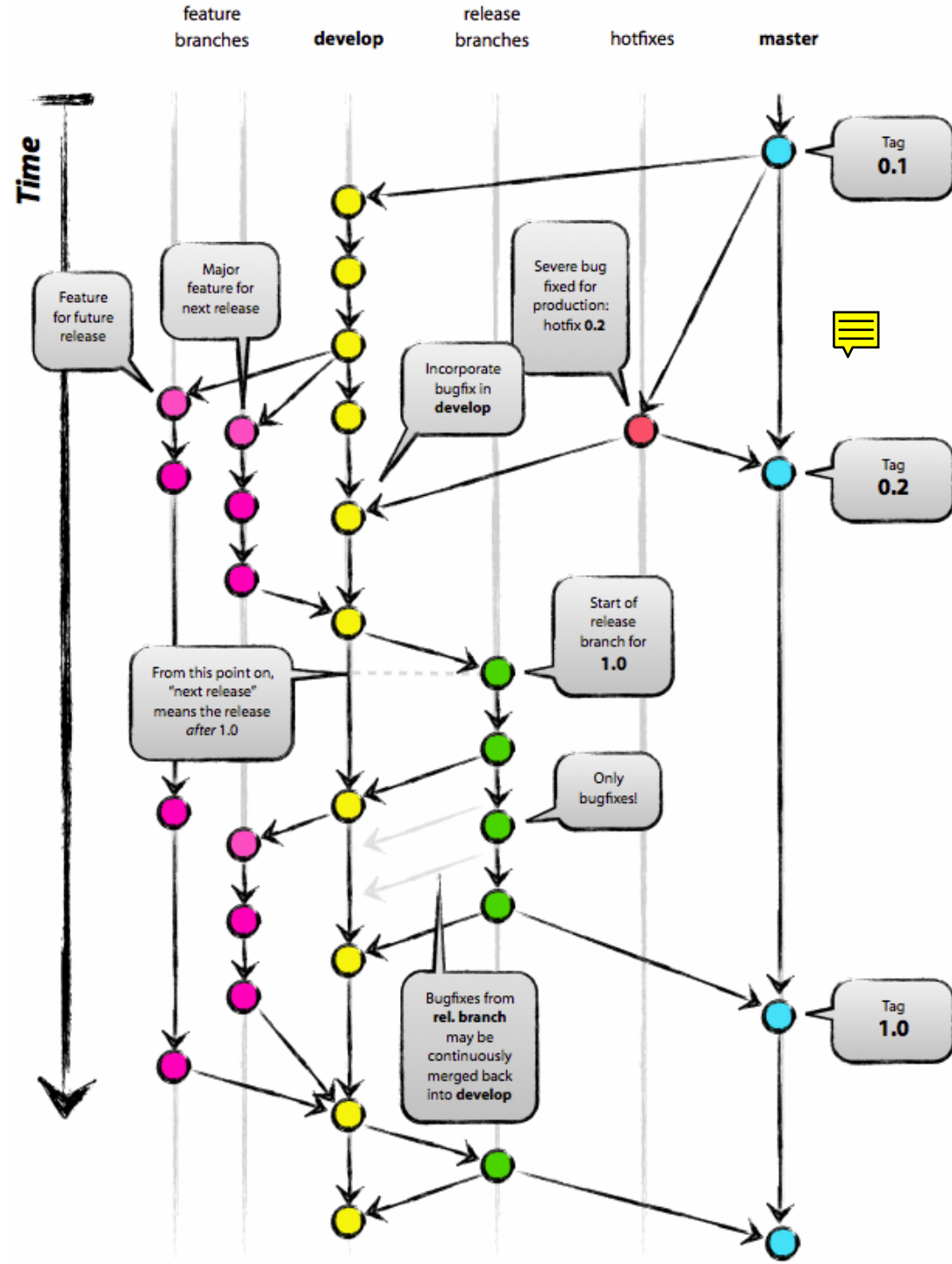
Types of Branches

- Master (Production)
- Development (Head) 
- Supporting branches
 - Feature or topic branches
 - Release or integration branches
 - Hotfix or quickfix branches

Branching And Process

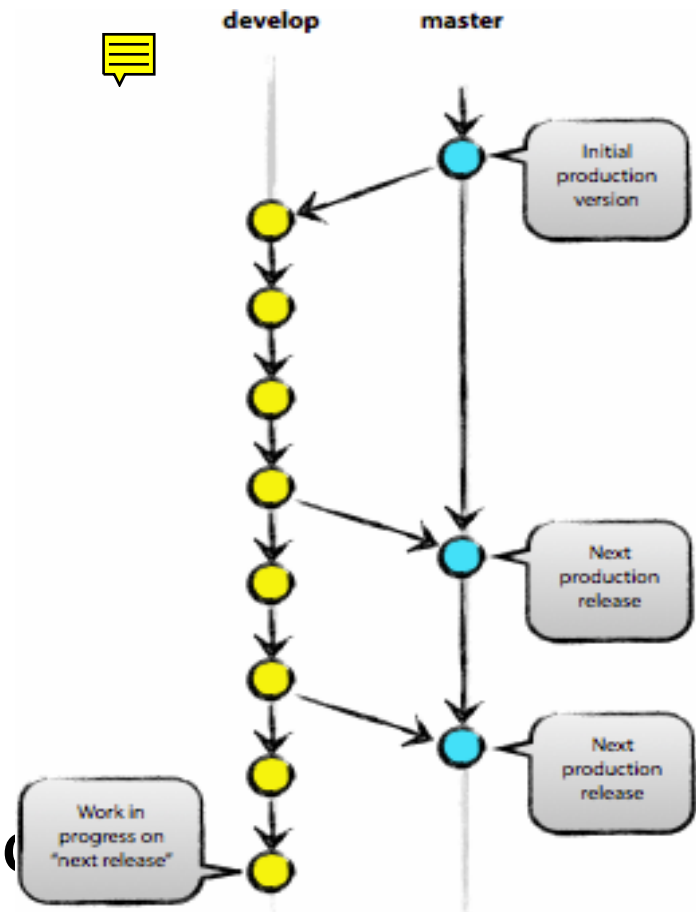
From:

<http://nvie.com/archives/323>



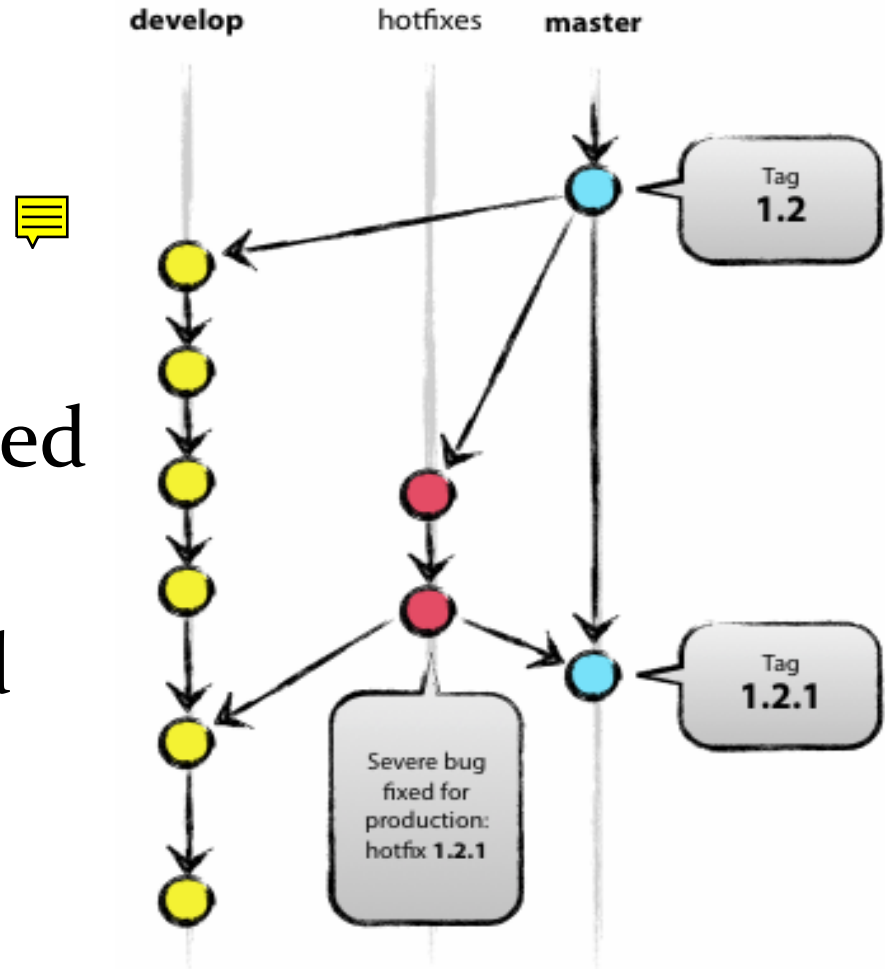
Development and Stable

- Devel branch has the current development integrated into it (e.g., head)
- The release branch has perfected changes and tagged releases
- Can be further subdivided



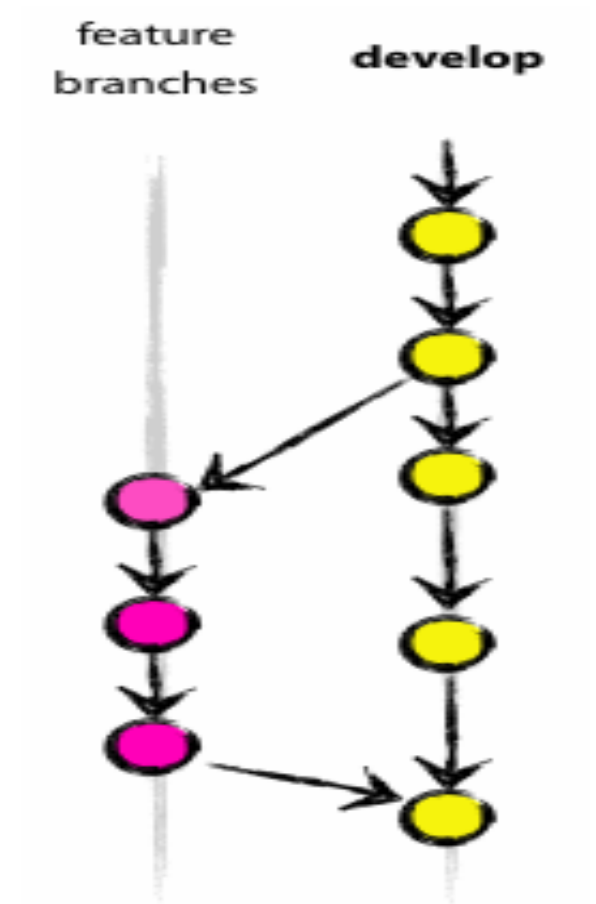
Hot fixes

- Branch from master
- Critical fix in a production system
- Can create a backported fix
- And is usually merged into current development



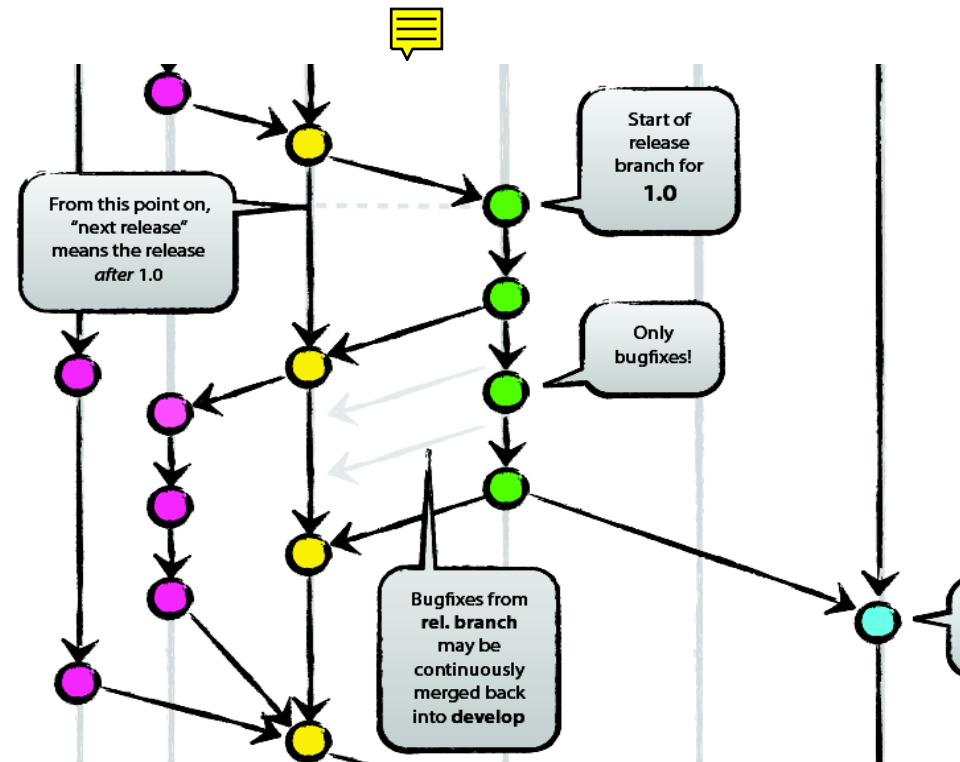
Feature branches

- Branch from and into develop
- AKA: topic branch
- Develop new and distant features
- Target release may be unknown
- Lasts until feature is finished or dies



Release Branch

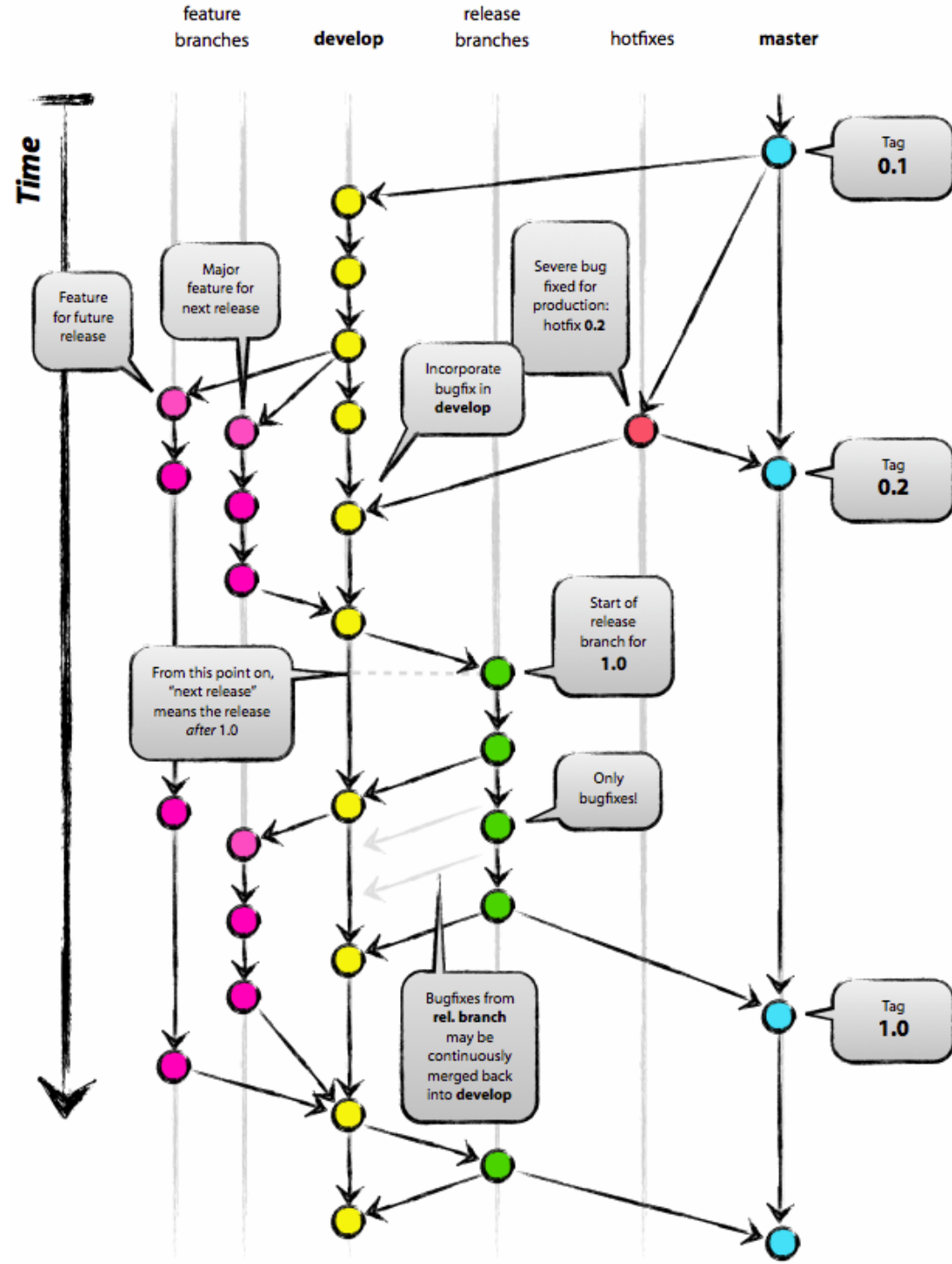
- From Development
- Stabilization
 - Stable feature integration
 - Bug fixes
 - Code freeze
 - Release number
- But development can continue on other branch for next release



Branching And Process

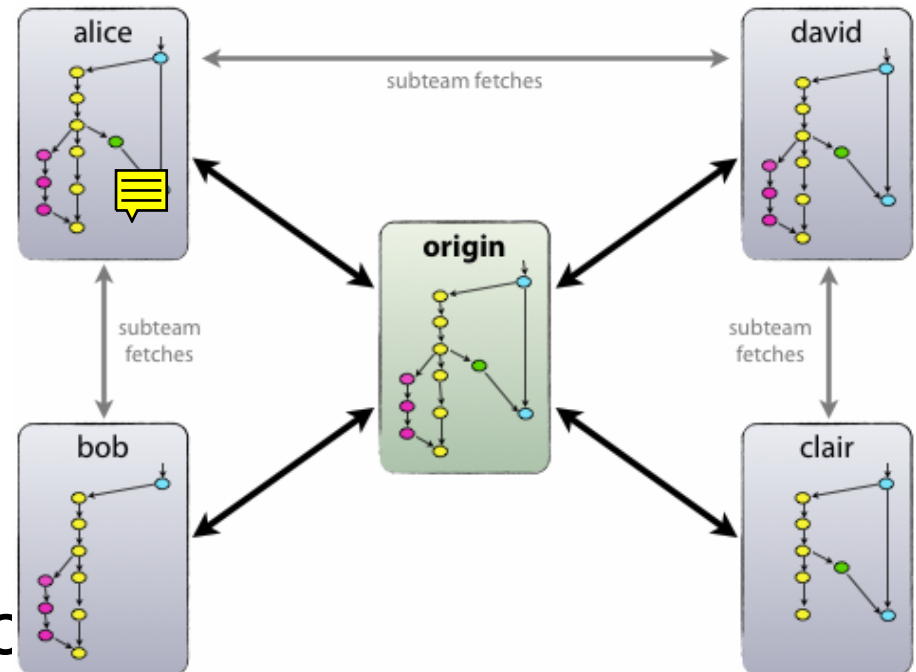
From:

<http://nvie.com/archives/323>



Mixing Centralized and Decentralized

- Central repository for integration and final changes (origin)
- Subteams
 - Sandboxes
 - Peer sharing




No repo is inherently more important
are socially more important

Sharing changes (patches)

- Directly through repositories
 - Difficult to perform review and discuss
- In a bug database
 - Often for outside users and developers only
- On a mailing list
 - 66% of threaded discussions on LKML have a patch
 - Difficult to track, often ignored, resend
- A patch is a diff and a changelog

Patch, Commit, Changeset



- Changes are “**small, independent, and complete**”
- They have: 
 - One line description of change
 - **Change log** that describes why and how a change was made (evolution)
 - The list of changed files
 - The **diffs** associated with the files
 - Bug, mailing list, and other linked artefacts
 - Credits, reviewers, sign-offs, etc

Bad Commit

r6228 | jrandom | 2004-06-30 22:13:07 -0500 (Wed, 30 Jun 2004) | 8 lines

Fix Issue #1729: Make indexing gracefully warn the user when a file is changing as it is being indexed.

- * ui/repl.py
 (ChangingFile): New exception class.
 (DoIndex): Handle new exception.
- * indexer/index.py
 (FollowStream): Raise new exception if file changes during indexing.
 (BuildDir): Unrelatedly, remove some obsolete comments, reformat some code, and fix the error check when creating a directory.

Other unrelated cleanups:

- * www/index.html: Fix some typos, set next release date.
-

Reading a Unified Diff

- Most common format
- --- /path/to/original_file
- +++ /path/to/new_file
- Has three lines of context
- New lines and remove lines are interleaved
 - + new line
 - - remove lined
- @@ -old start, old lines +new start, new lines
@@


Reading a Unified diff

```
--- a/arch/mn10300/kernel/time.c
+++ b/arch/mn10300/kernel/time.c
@@ -1,6 +1,6 @@
/* MN10300 Low level time management
 *
 * Copyright (C) 2007 Red Hat, Inc. All Rights Reserved.
+ * Copyright (C) 2007-2008 Red Hat, Inc. All Rights Reserved.
 * Written by David Howells (dhowells@redhat.com)
 * - Derived from arch/i386/kernel/time.c
 *
@@ -16,6 +16,7 @@
#include <linux/init.h>
#include <linux/smp.h>
#include <linux/profile.h>
+#include <linux/cnt32_to_63.h>
#include <asm/irq.h>
#include <asm/div64.h>
#include <asm/processor.h>
```

Patchset


- A series of commits that implement a feature or fix a bug
 - Independent but interrelated changes
 - Usual kept on a feature or topic branch
- Usually sent as an email for review, for example:
 - Patch 0/3: Fixing and combining foobar with bar
 - Patch 1/3: Fix of foobar
 - Patch 2/3: Integrate existing bar with foobar
 - Patch 3/3: Update documentation on bar

Example of patchset


- <http://lkml.org/lkml/2008/5/27/278>
- This is how the changes you submit to the project should look
 - Patchset 0/n (description of problem, interrelationship of commits, tests, system evolution, discussion with other, etc)
 - Patchset 1/n (first changeset)
 - Patchset 2/n (second changeset)
 - ... 
 - Patchset n/n (final changeset, probably update docs)
- You can also make a make a GitHub pull request

Managing a Release

Types of Releases

- Analogy (road maintenance)
- Shut down highway
 - Convenient for workers 
 - All branches frozen
- Shut down one lane
 - Convenient for others
 - Release branch frozen, others are open
- How does distributed VC affect the process?

Release Numbering


- Major . Minor . Micro
 - 2.6.34 
 - 3.11
- Major = version = series
- Minor = new releases of this series
- Micro = security fixes etc
- Others?
 - Backport number

Releasing Candidates

- Indicates a certain level of testing and assurance
- Alpha
 - Devs only (2.6-alpha)
- Beta
 - Larger community (2.6-beta)
- Can also use RC#
 - Increment number as release stabilizes (2.6-rc3)



Major, Minor, Micro


- Micro: same minor series
 - Forward- and backward-compatible
 - Small changes: bug fixes only or very small enhancements to existing features
 - No new features 
- Minor number: same major series
 - Backward-compatible,
 - Some new features
- Major number: new series
 - Not backwards or forwards compatible
 - New features or set of features

Release early, release often

- The longer interval between releases the more likely devs will want to push their latest unstable code into the release line



Stabilizing a Release

- Policy of what is included
 - Minor bug fixes
 - Doc updates etc
- Who decides
 - Release owner (dictator)
 - Voting by release group
- Release manager
 - Make the release go smoothly
 - Tracks unreviewed changes, etc

Summary

- Definitions
 - Commit, repository, branch, patchset, etc
- Good and bad isolation
- Working off Head vs off Bases
- Branches
 - CVC vs DVC
 - Types
 - Master, Devel, Feature, Release, Fixes
- Submitting changes: Commits, Branches, Patchsets
- Managing a release

References, acknowledgements, and readings

- Some material reused from Dr. Bull
- Required Reading
 - Chapter 7 of *Producing OSS*
 - <http://nvie.com/posts/a-successful-git-branching-model/>