Version Control Systems

An Introduction

J.R. Mauro
jrm8005@cs.rit.edu
www.cs.rit.edu/~jrm8005

Department of Computer Science
Rochester Institute of Technology

November 19, 2008

Agenda

Basic Questions

- VCS? SCM? What are we talking about? What does it all mean?

Basic Questions

- VCS? SCM? What are we talking about? What does it all mean?
  - VCS stands for Version Control System, though this isn't really what we're going to talk about
  - SCM stands for either "Software Configuration Management", or "Source Code Management", or "Software Change Management", or or. . .

Basic Questions

- VCS? SCM? What are we talking about? What does it all mean?
    - VCS stands for Version Control System, though this isn't really what we're going to talk about
    - SCM stands for either "Software Configuration Management", or "Source Code Management", or "Software Change Management", or or...
    - A system for managing multiple revisions of related source code files that comprise a single module or project, linearly and/or concurrently, which offers tools and strategies for managing differences between versions, concurrent development by multiple people, "branches", etc.

Basic Questions

- VCS? SCM? What are we talking about? What does it all mean?
  - VCS stands for Version Control System, though this isn't really what we're going to talk about
  - SCM stands for either "Software Configuration Management", or "Source Code Management", or "Software Change Management", or or...
  - A system for managing multiple revisions of related source code files that comprise a single module or project, linearly and/or concurrently, which offers tools and strategies for managing differences between versions, concurrent development by multiple people, "branches", etc.
- What is the Difference between SCM and Revision/Version Control?

Basic Questions

- VCS? SCM? What are we talking about? What does it all mean?
    - VCS stands for Version Control System, though this isn't really what we're going to talk about
    - SCM stands for either "Software Configuration Management", or "Source Code Management", or "Software Change Management", or or...
    - A system for managing multiple revisions of related source code files that comprise a single module or project, linearly and/or concurrently, which offers tools and strategies for managing differences between versions, concurrent development by multiple people, "branches", etc.
- What is the Difference between SCM and Revision/Version Control?
    - Revision control is strictly linear
    - Revision control has no capabilities for multiple developers

Why It’s Needed

- SCM is something you need whether you’re a programmer alone or on a team

Why It's Needed

- SCM is something you need whether you're a programmer alone or on a team
- SCM is something you need whether you're a programmer or not!

Why It's Needed

- SCM is something you need whether you're a programmer alone or on a team
- SCM is something you need whether you're a programmer or not!
- Why is Source Code Management important?

Why It's Needed

- SCM is something you need whether you're a programmer alone or on a team
- SCM is something you need whether you're a programmer or not!
- Why is Source Code Management important?
  - Safety and protection, from yourself, your tools, and others
  - "Sandboxing" new idea
  - Finding regressions
  - Long-term storage
  - Multiple people working at once

Ok, Tell Me More

- There are two main kinds of SCM
  - Centralized and Distributed
  - Centralized came first
  - Distributed is widely being recognized as the “Right Thing”
  - Despite its recent popularity, distributed has not displaced centralized, especially in corporate environments and in projects with extensive history or little concurrent development

Ok, Tell Me More

- There are two main kinds of SCM
  - Centralized and Distributed
  - Centralized came first
  - Distributed is widely being recognized as the “Right Thing”
  - Despite its recent popularity, distributed has not displaced centralized, especially in corporate environments and in projects with extensive history or little concurrent development
- Literally dozens of solutions commercial and free
- Various languages and platforms, C, C++, Python, even Haskell!
- Different ideas of concurrency and strategies for conflict resolution

Phraseology

repository	All the data files needed for the SCM to represent a project and function on it
commit, changeset	A single unit of work submitted to a repository
submit, commit (v)	To send one's changes to a repository
checkout	The act of obtaining a copy from a repository
branch	A copy of the codebase used for implementing new features or maintaining old releases.
master, trunk	The main branch of a project
merge	The process of combining two branches, done by the SCM software occasionally with human help
parent	The immediately previous commit. Also, the branch from which the branch in question was created.
working copy	A checkout from a repository, the workspace in which a programmer makes changes

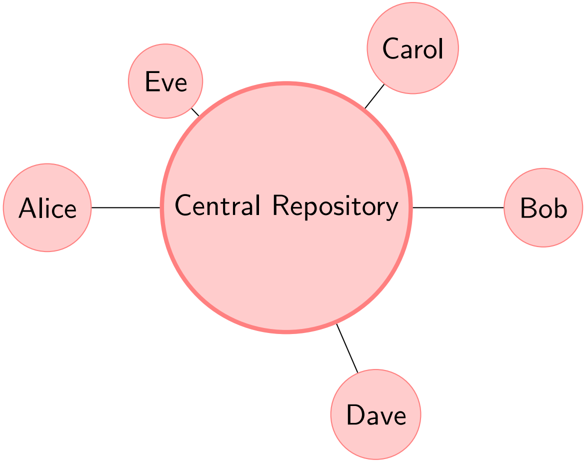
The Process

- Developers make their changes to the codebase, “committing” every now and then to the repository
- The SCM software handles these additions, creating changesets and storing them in a space-efficient manner relative to older versions
- This essentially introduces another step, one in which the developer stops coding, and takes time to submit, giving their changeset a description
- The setup of an SCMs repository can be complex, especially with the commonly-used “classic” tools like CVS and SVN
- This concept introduces quite a bit of overhead, but it will be shown to be worth it

SCM Timeline

1972	SCCS, originally released for the IBM System/370 computer, is ported to UNIX
1980s	RCS released, becomes part of the GNU project
June, 1986	Dick Grune releases CVS, a tool he developed to ease working with students on the Amsterdam Compiler Kit. CVS introduces branching
2000	CollabNet tries to address CVS's shortcomings with the release of Subversion
2002	Linux kernel adopts BitKeeper for revision control
2005	Git and Mercurial appear, interest in distributed SCMs skyrockets

Typical Centralized “Code Flow”



Workflow in Traditional SCMs

- Developers initialize repository, grant commit access
- Initial files created, bugtracker set up, etc.
- Developers begin coding
  - In theory, they commit small, logically coherent changes which at any point result in a stable system that builds and runs
  - In practice, developers are very bad at this
- Merge conflicts are dealt with when necessary. There may be a "Merge Guy"
- At some point, the source is frozen for release. A "maintenance branch" is set up while development continues on *master*

And this has more or less worked for 20 years. . .

Centralized is Bad

- One point of failure
- Too many jobs are done on the server
- No offline committing
- No built-in backup strategy
- No easy restore
- No content verification
- Notion of “commit access”
- Lack of future-proof concurrency designs from the 80’s and early 90’s

Git is Born

- In the early 2000s, the Linux kernel decides to use Bitkeeper, a proprietary SCM
  - There's tension over this because it's not free software, but BK was the only distributed SCM
- In 2005, various issues prevented the Linux Kernel developers from using BK, prompting Torvalds set out to find a good replacement
- Torvalds's investigations find that the good alternatives all had one problem: they were too slow
- Not one to be put off easily, Torvalds prototypes a "log Linus' state" kind of user-space filesystem, which allows him to do merges more quickly
- Development gathers, improving it into a sort of version control framework around which a usable SCM workflow could develop

Distributed is Good

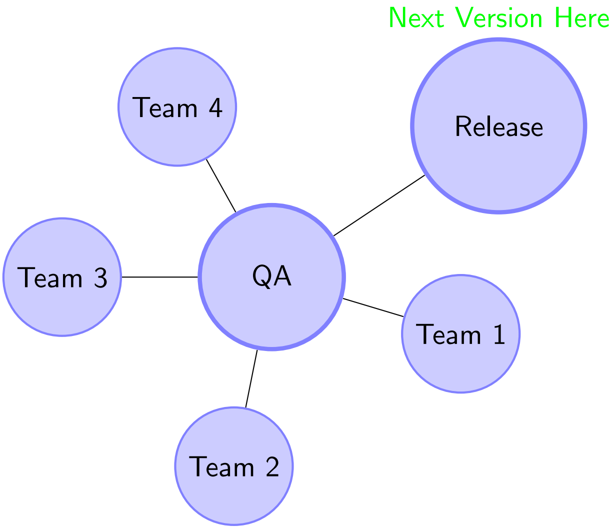
Contrasted with the pitfalls of centralized tools:

- No single point of failure
- Backups with distributed SCMs are simple: every single checkout is a working backup with the entire project history
- Many things, like diffs, are done locally. Only pushing/pulling puts strain on a non-local computer
- Restore is simple, as a client checkout has everything needed to act as a repository
- Content is verified with SHA1 hashes of file contents

Distributed is Good, continued

- No such thing as commit access. Everyone is free to clone and develop.
- Offline commits supported
- Concurrency is key to distribution, so it is well supported
- Commits are atomic, merging is easy, and better tools exist to help.
- Scalability and performance are better

Typical Distributed “Code Flow”



Distributed SCM Workflow

- Developers initialize repository. No need for commit access
- Initial files created, bugtracker set up, etc.
- Developers clone repository, possibly creating several "topic branches", begin coding
  - Developers can adopt any style they want, as there are many tools to amend commits.
  - Hopefully they approach the ideal commit strategy with the help of the tools
- Every developer merges, since they have their own branches which aren't controlled (or even seen, most of the time) by others.
- At some point, the source is frozen for release. A "maintenance branch" is set up while development continues on *master*

Key Git Concepts

The Index	A staging ground for the next commit
Interactive commits	Before publishing their work, developers are free to reorder their commits, move their changes from one commit to another, and squash several commits into one.
Rebasing	Developers can make a series of commits interleaved with upstream commits appear to happen linearly at the end of the latest upstream changeset
Topic Branches	Anyone can create as many branches as they want, so often developers have a branch for every sub-project they work on.

This awesome power has led to Git being described as a set of tools for “creating your own distributed SCM workflow”

Setting Up A Repo

1. Create a directory with your initial files
2. Run *git init*, followed by *git add .* and *git commit*
3. Configure network access with *git daemon* and *git instaweb*
4. Alternatively, get free git hosting on **Github**, **git.or.cz**, or **Gitorious**
5. Start coding!

If you're not starting your own project, you can run *git clone* $\langle$*repo*$\rangle$ to pull someone's code and start working

Day-to-Day Git

- Run *git pull* to receive changes from your parent and peer repositories
- Edit code, compile, test. . .
- Run *git add* on the files you've altered whose changes you want to appear in the next commit

Day-to-Day Git

- Run *git pull* to receive changes from your parent and peer repositories
- Edit code, compile, test. . .
- Run *git add* on the files you've altered whose changes you want to appear in the next commit
  - If you want CVS/SVN/P4 behavior, run *git commit -a*

Day-to-Day Git

- Run *git pull* to receive changes from your parent and peer repositories
- Edit code, compile, test. . .
- Run *git add* on the files you've altered whose changes you want to appear in the next commit
  - If you want CVS/SVN/P4 behavior, run *git commit -a*
- Create a branch for your new idea with *git branch* ⟨*name*⟩
- Checkout your maintenance branch to fix a bug with *git checkout -b* ⟨*branch1*⟩
- See what you did with *git diff* or *git status*
- Reorder all of your changes with *git rebase*
- Merge your fixes into your *master* branch with *git merge* ⟨*branch1*⟩ ⟨*branch2*⟩

Working with Others

- Use *git pull* and *git push* to share changes
- Check your mail for patches, apply them with *git apply* for a diff or *git am* to take them directly from an mbox
- Git will automerge for you, but it may need help. Use *git mergetool* to do this pleasantly
- Use *git log* to see what's happened and *git blame* to see line for line who changed file content
- *git cherrypick* can be useful to pull one select change from another branch
- Use *git format-patch* to send your changes out as patchsets, or use *git push* to send them to a repository
- Tag and PGP sign a release with *git tag*, and format a source code tarball or zip with *git archive*

Potpourri

- Add hooks to trigger before/after many events by placing them in *.git/hooks*
- Look on the web for user-contributed scripts like *git forest* and *git split*
- Import entire histories from competitors like CVS, SVN, P4, BZR, more!
  - You can even interact bidirectionally with any of these and no one will be the wiser
- Screw up a commit message or forget to add a file? Run *git commit –amend*
- Run *git commit –interactive* to select pieces of code to put into the index
- Completely change the course of your commit history with *git rebase -i* ⟨*ref*⟩

Recap: Why DVCS over CVCS?

Distribution offers:

- No single point of failure
- Easy branching and merging
- Easy collaboration with others
- Less administrative overhead
- Verification of content
- Easy backups
- High performance

Why Git instead of $SCM?

Some features Git offers that its competitors may not:

- The index
- Most allowable workflows
- Unparalleled speed (written in pure Linus-y C)
- Collection of small tools that can be scripted
- Possibly the best rebase support
- Best repository size after packing
- Fanboy Factor: Only SCM written by the Great Linus

Views

- Interest in distributed SCMs has exploded
- Because of high quality of merging and fault tolerance, the distributed model is increasingly being considered the best way to handle development
- Community projects benefit greatly as anyone can easily contribute without having to deal with user accounts
- The ease of distributed SCM merges encourages branches, which helps improve process, leading to higher quality code
- Many projects, including Mozilla, X.org, Wine, MoinMoin, Java, Open Solaris, and most recently Qt use DVCS, with many others considering or in the process of transitioning their SCM

Questions?

Questions?

About:
This presentation was made with LaTeX Beamer using the Frankfurt theme.

It was presented with **Accentuate**, a free slideshow program maintained by the author. See `www.cs.rit.edu/~jrm8005/accentuate.html` for more information.