

“Next Generation” Version Control Systems

...or why bzr, hg, and git rock so hard

S. Vilain

Catalyst IT (NZ) Limited

Open Source Conference, 2007

Outline

Historical Context

Distributed >> Centralised

Benefits of Distribution

The Model of Distribution

Revision Data Warehousing

Stable Development Model

The Variety of NG VCS Experience

The Players

Differentiating Factors

Pragmatic Concerns

Outline

Historical Context

Distributed >> Centralised

Benefits of Distribution

The Model of Distribution

Revision Data Warehousing

Stable Development Model

The Variety of NG VCS Experience

The Players

Differentiating Factors

Pragmatic Concerns

Outline

Historical Context

Distributed >> Centralised

- Benefits of Distribution

- The Model of Distribution

- Revision Data Warehousing

- Stable Development Model

The Variety of NG VCS Experience

- The Players

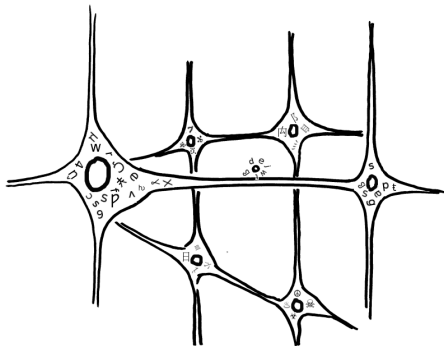
- Differentiating Factors

- Pragmatic Concerns

Neural Networks

c. 1bn. BCE-today

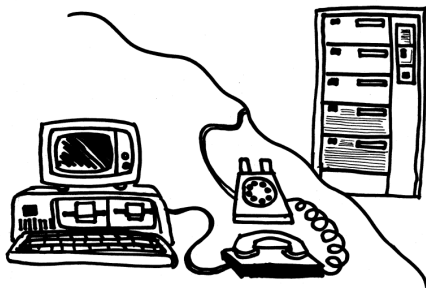
- ▶ Transmission of references to information
- ▶ Distributed in clusters (organisms)
- ▶ Generally support forking well
- ▶ Facilitated development of Mathematics and Computer Science



Detached Repository - Concurrent Versions System

1991-c.2001

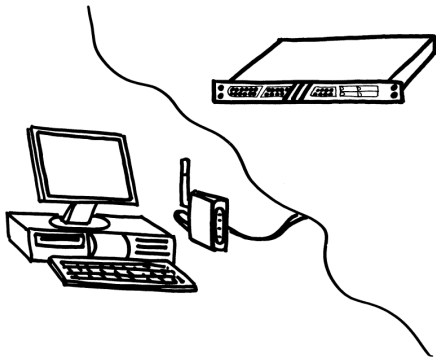
- ▶ Shell scripts to separate source from checkout
- ▶ Network separation via rsh
- ▶ Used branching support in RCS for concurrent development



Sub-Version System

2001-

- ▶ C programs to separate source from checkout
- ▶ Network separation via binary protocol, WebDAV
- ▶ Flattened branching, some copy efficiency, new “dimension” of properties



Distributed Development - Patch-based systems

from 1985

- ▶ patch: automatic application of context diffs
- ▶ Unified diffs - allow changes to be reviewed
- ▶ “tags” simply snapshots of source
- ▶ many tools based on patches - arch, bazaar, darcs



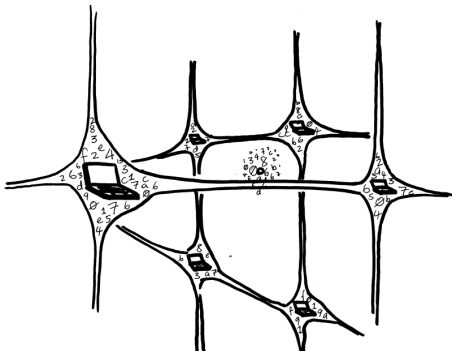
IM IN
YR PAST

WRTN
YR
TOOLZ

“Next Generation” tools

c.2002-

- ▶ Fully distributed
- ▶ Every revision trace-able
- ▶ Efficient packs/bundles (sets of revisions)
- ▶ Complete, uniform distribution of history
- ▶ Many peripheral benefits



Outline

Historical Context

Distributed >> Centralised

Benefits of Distribution

The Model of Distribution

Revision Data Warehousing

Stable Development Model

The Variety of NG VCS Experience

The Players

Differentiating Factors

Pragmatic Concerns

Distribution Benefits 1 of many

Central point of failure

Central point of failure:

- ▶ centralisation requires a “master” to, at the very least, assign commit IDs
- ▶ decentralisation assigns commit IDs in unique ways (content hashing, UUIDs)

So:

- ▶ Central servers become points of failure (for the services they provide) and contention. ie, your server goes down, people are interrupted
- ▶ collaboration between disconnected people impeded

Distribution Benefits 1 of many

Central point of failure

Central point of failure:

- ▶ centralisation requires a “master” to, at the very least, assign commit IDs
- ▶ decentralisation assigns commit IDs in unique ways (content hashing, UUIDs)

So:

- ▶ Central servers become points of failure (for the services they provide) and contention. ie, your server goes down, people are interrupted
- ▶ collaboration between disconnected people impeded

Distribution Benefits 2 of many

transactions or atomic commits?

So, your Centralised VCS gives you “Atomic Updates”

Unix guarantees write ordering on filehandles, but that does not make it a database.

“A” is only one letter out of “ACID”.

So,

- ▶ centralisation is inherently non-transactional - “dirty read” - changes all in the same place
- ▶ decentralisation is inherently transactional - “consistent read” - your changes don't affect others

Distribution Benefits 2 of many

transactions or atomic commits?

So, your Centralised VCS gives you “Atomic Updates”

Unix guarantees write ordering on filehandles, but that does not make it a database.

“A” is only one letter out of “ACID”.

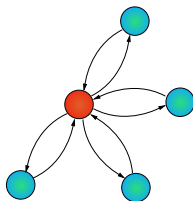
So,

- ▶ centralisation is inherently non-transactional - “dirty read” - changes all in the same place
- ▶ decentralisation is inherently transactional - “consistent read” - your changes don't affect others

Distribution Benefits 3 of many

any to any merge pattern

- ▶ centralisation requires the “Star” pattern
 - ▶ one big cluster of development
- ▶ decentralisation makes such constructs optional or notional
 - ▶ self-forming clusters of development



Outline

Historical Context

Distributed >> Centralised

Benefits of Distribution

The Model of Distribution

Revision Data Warehousing

Stable Development Model

The Variety of NG VCS Experience

The Players

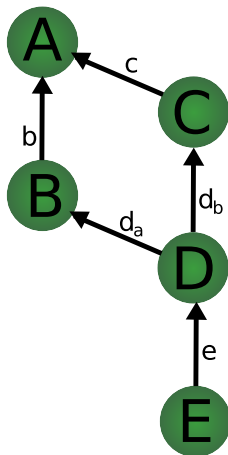
Differentiating Factors

Pragmatic Concerns

Revision Model requirements for Distribution

The Revision DAG

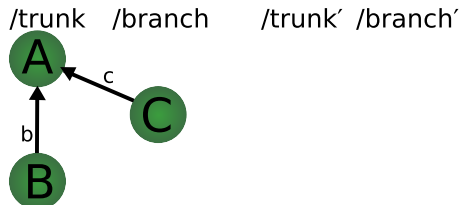
- ▶ must represent merges to work
- ▶ versions must not change by location
- ▶ branching is in the direction of **changes**



When Revision Models go Wrong #1

merge tracking can refer to other repositories

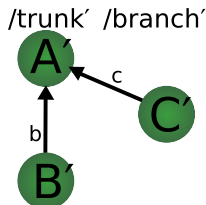
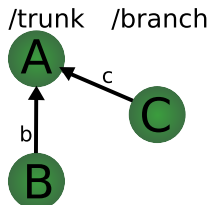
- ▶ initial state
- ▶ sync from mirror
- ▶ perform Merge
- ▶ another change
- ▶ push - whoops
- ▶ push



When Revision Models go Wrong #1

merge tracking can refer to other repositories

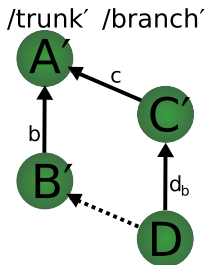
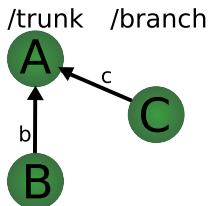
- ▶ initial state
- ▶ sync from mirror
- ▶ perform Merge
- ▶ another change
- ▶ push - whoops
- ▶ push



When Revision Models go Wrong #1

merge tracking can refer to other repositories

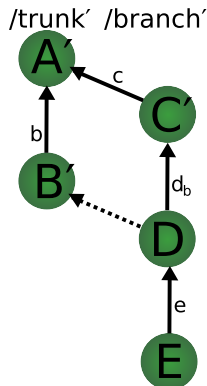
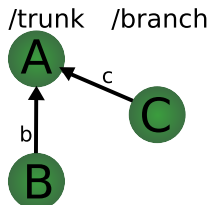
- ▶ initial state
- ▶ sync from mirror
- ▶ perform Merge
- ▶ another change
- ▶ push - whoops
- ▶ push



When Revision Models go Wrong #1

merge tracking can refer to other repositories

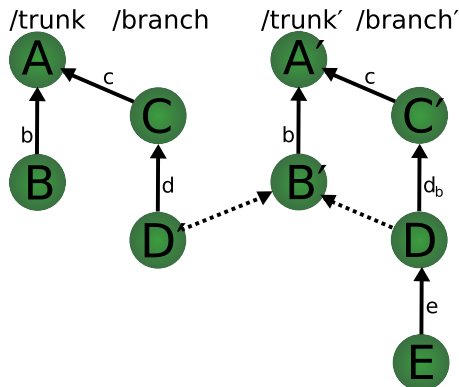
- ▶ initial state
- ▶ sync from mirror
- ▶ perform Merge
- ▶ another change
- ▶ push - whoops
- ▶ push



When Revision Models go Wrong #1

merge tracking can refer to other repositories

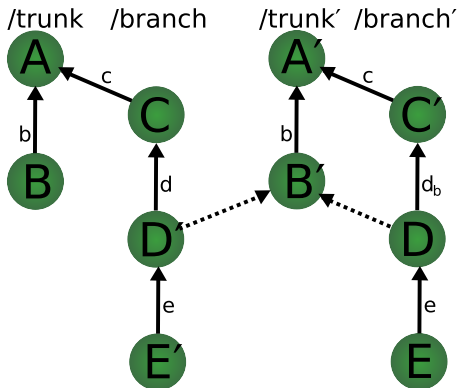
- ▶ initial state
- ▶ sync from mirror
- ▶ perform Merge
- ▶ another change
- ▶ push - whoops
- ▶ push



When Revision Models go Wrong #1

merge tracking can refer to other repositories

- ▶ initial state
- ▶ sync from mirror
- ▶ perform Merge
- ▶ another change
- ▶ push - whoops
- ▶ push



When Revision Models go Wrong #2

merge tracking within a repository

(as in SVN 1.5+)

- ▶ initial state
- ▶ make a change and new branch
- ▶ make a change to trunk
- ▶ merge trunk to branch
- ▶ merge branch to trunk. Note r4, r5 have (or should have) identical content and set of changes.
- ▶ what will merge from trunk to branch do?
- ▶ distributed model resists the problem

/trunk

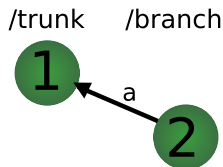
1

When Revision Models go Wrong #2

merge tracking within a repository

(as in SVN 1.5+)

- ▶ initial state
- ▶ make a change and new branch
- ▶ make a change to trunk
- ▶ merge trunk to branch
- ▶ merge branch to trunk. Note r4, r5 have (or should have) identical content and set of changes.
- ▶ what will merge from trunk to branch do?
- ▶ distributed model resists the problem

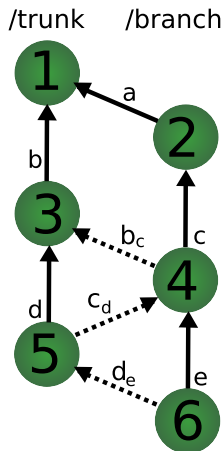


When Revision Models go Wrong #2

merge tracking within a repository

(as in SVN 1.5+)

- ▶ initial state
- ▶ make a change and new branch
- ▶ make a change to trunk
- ▶ merge trunk to branch
- ▶ merge branch to trunk. Note r4, r5 have (or should have) identical content and set of changes.
- ▶ what will merge from trunk to branch do?
- ▶ distributed model resists the problem

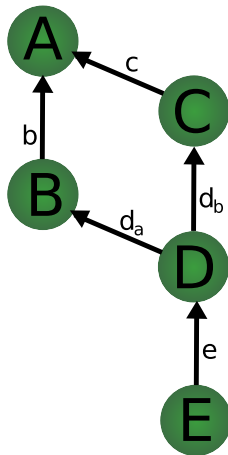


When Revision Models go Wrong #2

merge tracking within a repository

(as in SVN 1.5+)

- ▶ initial state
- ▶ make a change and new branch
- ▶ make a change to trunk
- ▶ merge trunk to branch
- ▶ merge branch to trunk. Note r4, r5 have (or should have) identical content and set of changes.
- ▶ what will merge from trunk to branch do?
- ▶ distributed model resists the problem



Outline

Historical Context

Distributed >> Centralised

Benefits of Distribution

The Model of Distribution

Revision Data Warehousing

Stable Development Model

The Variety of NG VCS Experience

The Players

Differentiating Factors

Pragmatic Concerns

Revision Data Warehouse

On-Line Analytical Processing of your source

- ▶ Goal: find more information on the reasons for changes
- ▶ Observation: 'log' is only one approach of many
- ▶ `git-log -S 'string'` : find changes that introduced "string" (also `hg grep`, `hgrep` plug-in for `bzr`)
- ▶ `git-annotate -C` : follows lines moving *between source files*
- ▶ visualization tools: `gitk`, `hgk`, `olive`, that allow advanced inspection of history

Outline

Historical Context

Distributed >> Centralised

Benefits of Distribution

The Model of Distribution

Revision Data Warehousing

Stable Development Model

The Variety of NG VCS Experience

The Players

Differentiating Factors

Pragmatic Concerns

Stable Development Model

It's not just a slogan

- ▶ Perhaps the major benefit of DSCM
- ▶ Using DSCM doesn't give you a stable development model for free
- ▶ This development practice *is* possible with SVK or even Subversion, in principle
- ▶ ...but with DSCM it's actually achievable and even common practice, because **bad changes are more easily dropped**

Stable Development Model

It's not just a slogan

- ▶ Perhaps the major benefit of DSCM
- ▶ Using DSCM doesn't give you a stable development model for free
- ▶ This development practice *is* possible with SVK or even Subversion, in principle
- ▶ ...but with DSCM it's actually achievable and even common practice, because **bad changes are more easily dropped**

Stable Development Model

It's not just a slogan

- ▶ Perhaps the major benefit of DSCM
- ▶ Using DSCM doesn't give you a stable development model for free
- ▶ This development practice *is* possible with SVK or even Subversion, in principle
- ▶ ...but with DSCM it's actually achievable and even common practice, because **bad changes are more easily dropped**

Stable Development Model

What you pay for what pay-off

Discipline:

- ▶ no single commit can break the (build/test suite/etc)
- ▶ every single commit is well described

Benefits:

- ▶ bisect: finding exactly which commit ruined your day, as every revision should build and work
- ▶ review: much easier for third parties to comment
- ▶ stability: if done right, even people tracking bleeding edge don't get put off working on the project by romping instability

Stable Development Model

What you pay for what pay-off

Discipline:

- ▶ no single commit can break the (build/test suite/etc)
- ▶ every single commit is well described

Benefits:

- ▶ bisect: finding exactly which commit ruined your day, as every revision should build and work
- ▶ review: much easier for third parties to comment
- ▶ stability: if done right, even people tracking bleeding edge don't get put off working on the project by romping instability

Outline

Historical Context

Distributed >> Centralised

Benefits of Distribution

The Model of Distribution

Revision Data Warehousing

Stable Development Model

The Variety of NG VCS Experience

The Players

Differentiating Factors

Pragmatic Concerns

Version Control: The Next Generation

Enterprise-ready

- ▶ **Bazaar-NG** (bzd) - python-based. Longest runner, not as fast as the others but still keeping pace with features.
- ▶ **git** - Unix-style CL-API to internals. Blinding fast at almost everything. Extremely active. Sports the “Content Hashed Filesystem” idea stolen from Monotone.
- ▶ **Mercurial** (hg) - python and C. Also extremely fast, with progress and features defying their relatively small community.

Version Control: The Next Generation

Enterprise-ready

- ▶ **Bazaar-NG** (bzd) - python-based. Longest runner, not as fast as the others but still keeping pace with features.
- ▶ **git** - Unix-style CL-API to internals. Blinding fast at almost everything. Extremely active. Sports the “Content Hashed Filesystem” idea stolen from Monotone.
- ▶ **Mercurial** (hg) - python and C. Also extremely fast, with progress and features defying their relatively small community.

Version Control: The Next Generation

Enterprise-ready

- ▶ **Bazaar-NG** (bzd) - python-based. Longest runner, not as fast as the others but still keeping pace with features.
- ▶ **git** - Unix-style CL-API to internals. Blinding fast at almost everything. Extremely active. Sports the “Content Hashed Filesystem” idea stolen from Monotone.
- ▶ **Mercurial** (hg) - python and C. Also extremely fast, with progress and features defying their relatively small community.

Outline

Historical Context

Distributed >> Centralised

Benefits of Distribution

The Model of Distribution

Revision Data Warehousing

Stable Development Model

The Variety of NG VCS Experience

The Players

Differentiating Factors

Pragmatic Concerns

NG Tools: Inodes vs Content

who do you trust?

- ▶ `git` - considers inode history uninteresting, derivable from the content. No conventions for explicitly recording inode movement history (renames etc)
 - ▶ **pros**: history mining more advanced by necessity, no scope for recording such information incorrectly.
 - ▶ **cons**: occasionally doesn't detect renames, any inode-based operation relatively slow
- ▶ `hg`, `bzr` - file-based backing store means that inode history is the primary approach
 - ▶ **pros**: “lossless” (or, GIGO if you prefer) storing of file history
 - ▶ **cons**: commit files the wrong way and you might not see the real history.

NG Tools: Inodes vs Content

who do you trust?

- ▶ `git` - considers inode history uninteresting, derivable from the content. No conventions for explicitly recording inode movement history (renames etc)
 - ▶ **pros**: history mining more advanced by necessity, no scope for recording such information incorrectly.
 - ▶ **cons**: occasionally doesn't detect renames, any inode-based operation relatively slow
- ▶ `hg`, `bzr` - file-based backing store means that inode history is the primary approach
 - ▶ **pros**: “lossless” (or, GIGO if you prefer) storing of file history
 - ▶ **cons**: commit files the wrong way and you might not see the real history.

NG Tools: UUID generation

random or just pseudo-random?

- ▶ hg, git - revision IDs checksum the content and revision history, therefore offer *lossless revisions*
- ▶ bzzr - patch IDs are purely UUIDs. Technically could therefore lose data back to the last signed tag (“testament”)

UUIDs:

- ▶ **pros:** when *cherry picking*, give you a token to refer to. Also, *partial imports* when dealing with foreign VCSes easier
- ▶ **cons:** such tokens don't guarantee anything about the delivered change

NG Tools: UUID generation

random or just pseudo-random?

- ▶ hg, git - revision IDs checksum the content and revision history, therefore offer *lossless revisions*
- ▶ bazaar - patch IDs are purely UUIDs. Technically could therefore lose data back to the last signed tag (“testament”)

UUIDs:

- ▶ **pros:** when *cherry picking*, give you a token to refer to. Also, *partial imports* when dealing with foreign VCSes easier
- ▶ **cons:** such tokens don't guarantee anything about the delivered change

NG Tools: UUID generation

random or just pseudo-random?

- ▶ hg, git - revision IDs checksum the content and revision history, therefore offer *lossless revisions*
- ▶ bazaar - patch IDs are purely UUIDs. Technically could therefore lose data back to the last signed tag (“testament”)

UUIDs:

- ▶ **pros:** when *cherry picking*, give you a token to refer to. Also, *partial imports* when dealing with foreign VCSes easier
- ▶ **cons:** such tokens don't guarantee anything about the delivered change

NG Tools: Lightweight Branches

- ▶ important to encourage frequent *topic branching*
- ▶ ideal: *every* bug, feature, etc can get its own branch from last stable release until it is fully reviewed and known to be good.
- ▶ `git` - virtually pioneered lightweight branches
- ▶ `hg` - now supports lightweight branches well, though repositories still accumulate deleted branches. Use `mq`
- ▶ `bzr` - not directly supported, so less efficient (but branching still common practice)

Advanced add-ons to manage refining changes - Stacked Git (`stg`) and `guilt` in `git` and Mercurial Queues (`mq`) in `hg`.
For `bzr` there is also Patch Queue Manager (PQM) - branch dashboard, and *rebase plug-in*

NG Tools: Lightweight Branches

- ▶ important to encourage frequent *topic branching*
- ▶ ideal: *every* bug, feature, etc can get its own branch from last stable release until it is fully reviewed and known to be good.
- ▶ `git` - virtually pioneered lightweight branches
- ▶ `hg` - now supports lightweight branches well, though repositories still accumulate deleted branches. Use `mq`
- ▶ `bzr` - not directly supported, so less efficient (but branching still common practice)

Advanced add-ons to manage refining changes - Stacked Git (`stg`) and `guilt` in `git` and Mercurial Queues (`mq`) in `hg`.
For `bzr` there is also Patch Queue Manager (PQM) - branch dashboard, and *rebase plug-in*

NG Tools: Lightweight Branches

- ▶ important to encourage frequent *topic branching*
- ▶ ideal: *every* bug, feature, etc can get its own branch from last stable release until it is fully reviewed and known to be good.
- ▶ git - virtually pioneered lightweight branches
- ▶ hg - now supports lightweight branches well, though repositories still accumulate deleted branches. Use `mq`
- ▶ bzz - not directly supported, so less efficient (but branching still common practice)

Advanced add-ons to manage refining changes - Stacked Git (`stg`) and `guilt` in git and Mercurial Queues (`mq`) in hg. For bzz there is also Patch Queue Manager (PQM) - branch dashboard, and *rebase plug-in*

Outline

Historical Context

Distributed >> Centralised

Benefits of Distribution

The Model of Distribution

Revision Data Warehousing

Stable Development Model

The Variety of NG VCS Experience

The Players

Differentiating Factors

Pragmatic Concerns

NG SCMs: ease of use

nanna-ready

- ▶ `bzr` and `hg` - ease of use and simplicity always considered a driving focus.
- ▶ `git` - “If you want a usability feature, go implement it you lazy user, this is Open Source, scratch your own itch would you?”

...however these days it's much of a muchness

...however also remember I prefer `git`, so I *would* say that

NG SCMs: ease of use

nanna-ready

- ▶ bazaar and hg - ease of use and simplicity always considered a driving focus.
- ▶ git - “If you want a usability feature, go implement it you lazy user, this is Open Source, scratch your own itch would you?”

...however these days it's much of a muchness

...however also remember I prefer git, so I *would* say that

NG SCMs: ease of use

nanna-ready

- ▶ bazaar and hg - ease of use and simplicity always considered a driving focus.
- ▶ git - “If you want a usability feature, go implement it you lazy user, this is Open Source, scratch your own itch would you?”

...however these days it's much of a muchness

...however also remember I prefer git, so I *would* say that

NG SCMs: ease of use

nanna-ready

- ▶ bazaar and hg - ease of use and simplicity always considered a driving focus.
- ▶ git - “If you want a usability feature, go implement it you lazy user, this is Open Source, scratch your own itch would you?”

...however these days it's much of a muchness

...however also remember I prefer git, so I *would* say that

NG SCMs: portability

- ▶ hg - performs well on Windows and Unix
- ▶ bzr - performs and *installs* well on both
- ▶ git - written by Linus Torvalds

NG SCMs: portability

- ▶ hg - performs well on Windows and Unix
- ▶ bazaar - performs and *installs* well on both
- ▶ git - written by Linus Torvalds

NG SCMs: portability

- ▶ hg - performs well on Windows and Unix
- ▶ bazaar - performs and *installs* well on both
- ▶ git - written by Linus Torvalds

NG SCMs: portability

- ▶ hg - performs well on Windows and Unix
- ▶ bazaar - performs and *installs* well on both
- ▶ git - written by Linus Torvalds

“You want it to work on Windows?
This is Open Source, scratch your
own itch would you you lazy
Windows user?!”

NG SCMs: portability

- ▶ hg - performs well on Windows and Unix
- ▶ bazaar - performs and *installs* well on both
- ▶ git - written by Linus Torvalds
- ▶ Cygwin works today (slowly)
- ▶ MinGW port shows promise
- ▶ Minimal pure-Java implementation (for Eclipse)
- ▶ C# .NET implementation underway by Mono crew

NG SCMs: speed

- ▶ `bzr` - not a primary focus. “Fast enough for most users”. Good to many thousands of changes.
- ▶ `hg` - optimised for the “cold cache” case. Very fast.
- ▶ `git` - optimises for the “warm cache” case. Very fast.

`git` and `hg` have certain operations one or the other is faster or slower at, but they are both far beyond the performance of virtually everything else.

NG SCMs: speed

- ▶ `bzr` - not a primary focus. “Fast enough for most users”. Good to many thousands of changes.
- ▶ `hg` - optimised for the “cold cache” case. Very fast.
- ▶ `git` - optimises for the “warm cache” case. Very fast.

`git` and `hg` have certain operations one or the other is faster or slower at, but they are both far beyond the performance of virtually everything else.

NG SCMs: speed

- ▶ `bzr` - not a primary focus. “Fast enough for most users”. Good to many thousands of changes.
- ▶ `hg` - optimised for the “cold cache” case. Very fast.
- ▶ `git` - optimises for the “warm cache” case. Very fast.

`git` and `hg` have certain operations one or the other is faster or slower at, but they are both far beyond the performance of virtually everything else.

NG SCMs: speed

- ▶ `bzr` - not a primary focus. “Fast enough for most users”. Good to many thousands of changes.
- ▶ `hg` - optimised for the “cold cache” case. Very fast.
- ▶ `git` - optimises for the “warm cache” case. Very fast.

`git` and `hg` have certain operations one or the other is faster or slower at, but they are both far beyond the performance of virtually everything else.

NG SCMs: repository size

- ▶ **bzr** - not a primary focus, though still quite efficient.
- ▶ **hg** - very tight packs and repositories. Can “repack” via `hg bundle`
- ▶ **git** - very tight packs and repositories. in principle more space efficient than `hg`, but only rarely borne out in practice. Typically 10 times smaller repositories than SVN fsfs.

Both `git` and `hg` often get the entire project history and head checkout into a space smaller than a single `svn HEAD` checkout. GCC is one extreme example of this - 1.1GB SVN head checkout, 280MB `git` repository (vs approx. 18GB for the full SVN repository).

NG SCMs: repository size

- ▶ `bzr` - not a primary focus, though still quite efficient.
- ▶ `hg` - very tight packs and repositories. Can “repack” via `hg bundle`
- ▶ `git` - very tight packs and repositories. in principle more space efficient than `hg`, but only rarely borne out in practice. Typically 10 times smaller repositories than SVN fsfs.

Both `git` and `hg` often get the entire project history and head checkout into a space smaller than a single `svn HEAD` checkout. GCC is one extreme example of this - 1.1GB SVN head checkout, 280MB `git` repository (vs approx. 18GB for the full SVN repository).

NG SCMs: repository size

- ▶ `bzr` - not a primary focus, though still quite efficient.
- ▶ `hg` - very tight packs and repositories. Can “repack” via `hg bundle`
- ▶ `git` - very tight packs and repositories. in principle more space efficient than `hg`, but only rarely borne out in practice. Typically 10 times smaller repositories than SVN fsfs.

Both `git` and `hg` often get the entire project history and head checkout into a space smaller than a single `svn HEAD` checkout. GCC is one extreme example of this - 1.1GB SVN head checkout, 280MB `git` repository (vs approx. 18GB for the full SVN repository).

NG SCMs: repository size

- ▶ `bzr` - not a primary focus, though still quite efficient.
- ▶ `hg` - very tight packs and repositories. Can “repack” via `hg bundle`
- ▶ `git` - very tight packs and repositories. in principle more space efficient than `hg`, but only rarely borne out in practice. Typically 10 times smaller repositories than SVN fsfs.

Both `git` and `hg` often get the entire project history and head checkout into a space smaller than a single `svn HEAD` checkout. GCC is one extreme example of this - 1.1GB SVN head checkout, 280MB `git` repository (vs approx. 18GB for the full SVN repository).

Summary

- ▶ The three NG tools covered differ chiefly in **efficiency** and **user interface**
- ▶ The **revision graph** concept is seen in all these tools.
- ▶ A **location independent revision model** is paramount to successfully achieving the **stable development model**

For Further Reading I



Various sections of relevance on Wikipedia articles

http://en.wikipedia.org/wiki/Comparison_of_revision_control_software

http://en.wikipedia.org/wiki/Git_%28software%29#References



S. Vilain.

An Introduction to git-svn for Subversion/SVK users and deserters (advocacy and limitations sections)

<http://utsl.gen.nz/talks/git-svn/intro.html#wtf-why>

<http://utsl.gen.nz/talks/git-svn/intro.html#sux>