Version Control – Patterns & Practices

Chris Oldwood

ACCU Conference 2014

@chrisoldwood / gort@cix.co.uk

Brief overview of the topic and myself – the 7 VCS used so far (different one each time), still many unused

Acts as a time-machine, and almost as contentious as the text editor

This talk tries to address the cargo cult behaviour around branching strategies

Quick walkthrough of the schedule (it generally follows the software development lifecycle)

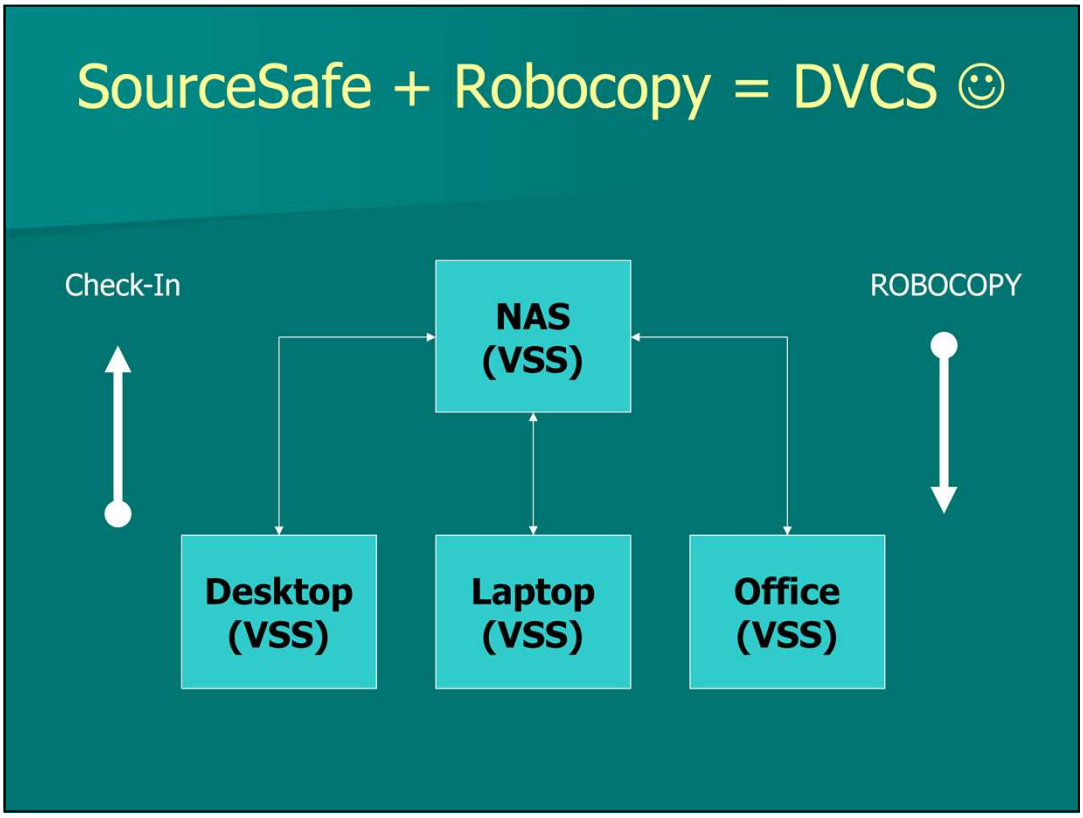
Architectures

Centralised versus distributed

Online versus self-hosted (dedicated VCS team)

Some have first class concepts for labels and branches, others it’s a convention

Shims muddy the waters, e.g. git-svn



Use local clones for history and as a local branch

Check-in (again) to master NAS (with proper comment)

Use robocopy to sync clones again

Not a sane setup!

Workspace

Private workspace – work in isolation and make your own mess.

Multiple workspaces are an alternative to switching the same workspace to different branches.

ClearCase – snapshot (local) vs. dynamic (network) views.

Ideally the workspace should contain everything to get started (e.g. build + unit tests).

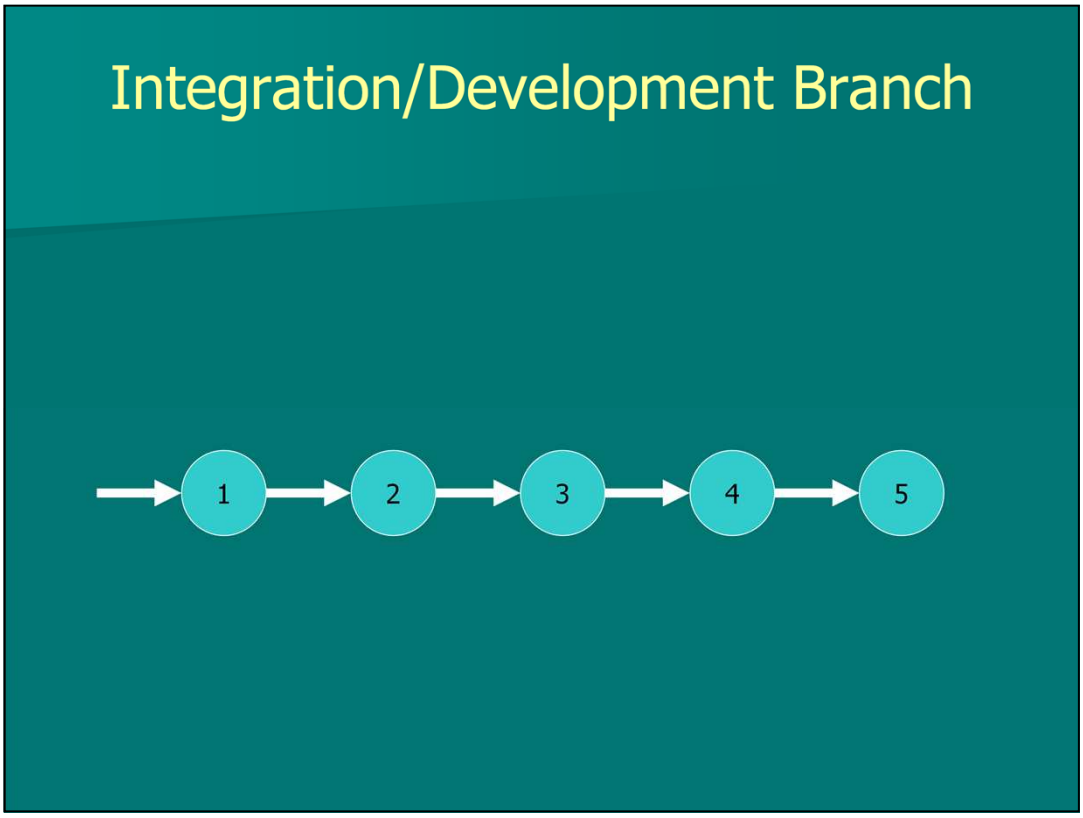
Usually workspace = branch, but can contain many branches (e.g. ClearCase config specs).

Branching

Contentions subject, highly dependent on the organisation (survival rules can determine the strategy).

Branches have policies, when a code change and policy are incompatible we can choose to branch.

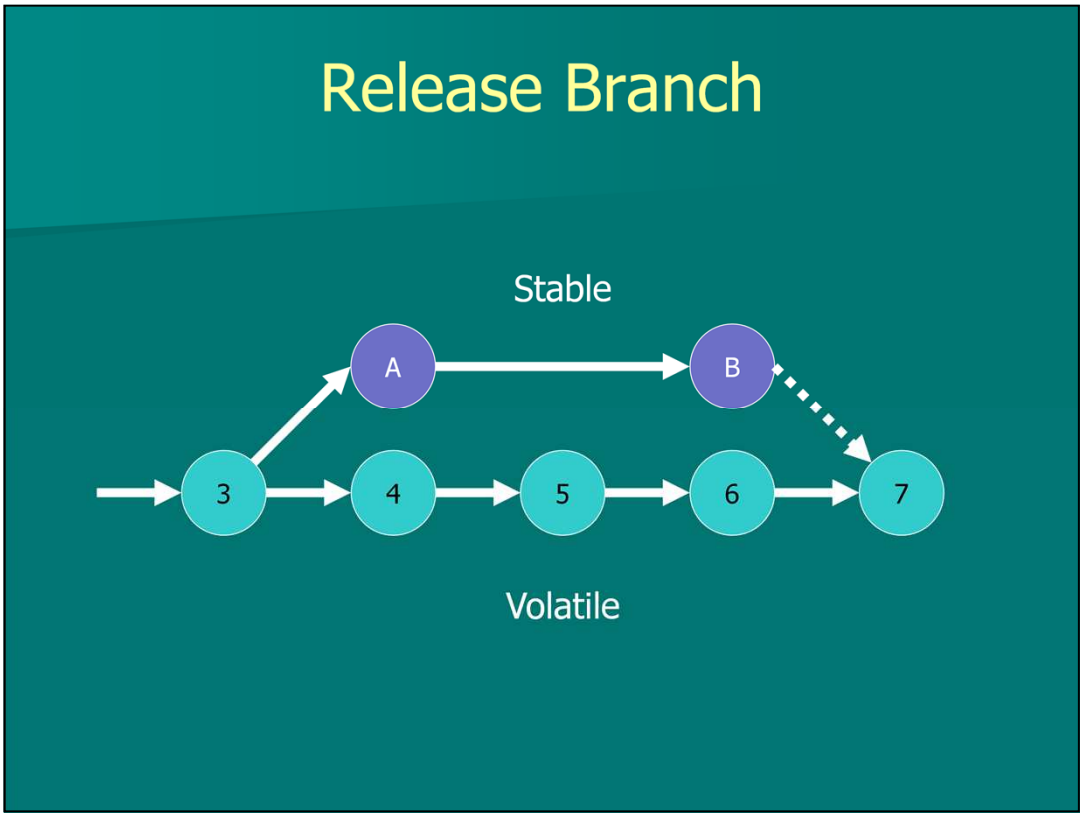
But branching is often a reaction to a weak development process – other ways to mitigate the risk.



The default branch when VCS doesn’t support branching.

Called different names - trunk/main/master.

A common “Enterprise” anti-pattern is one integration branch per project.

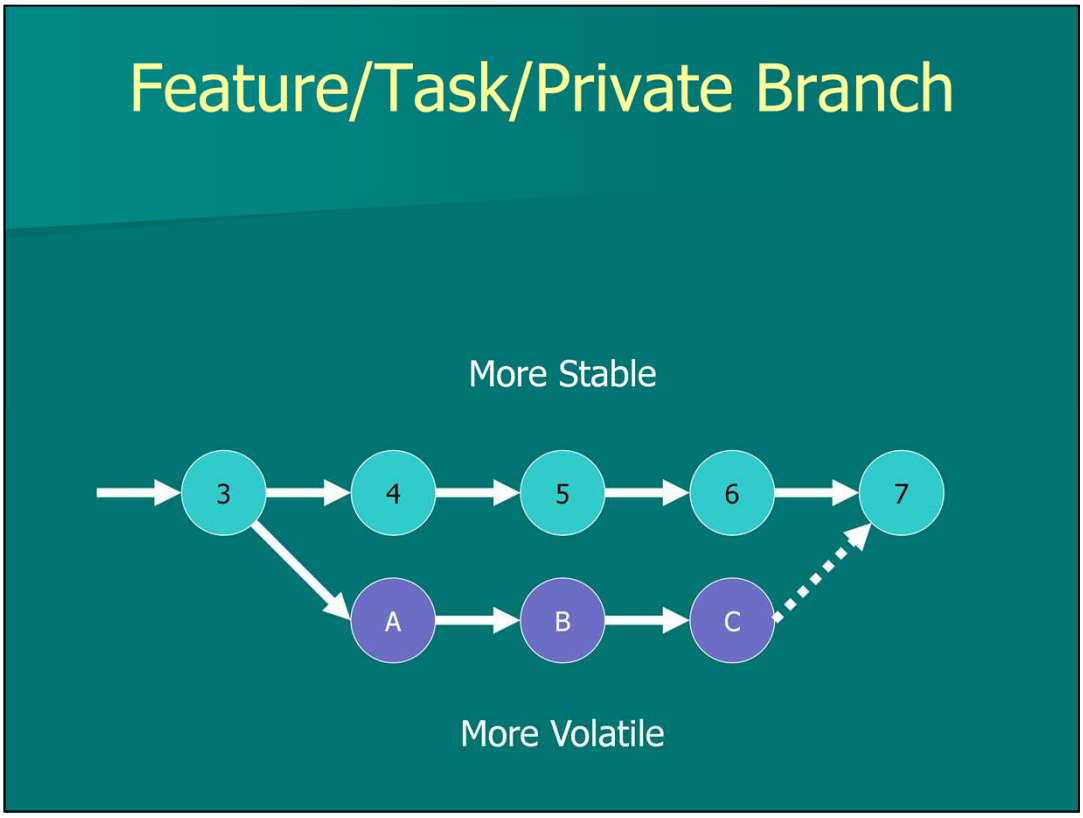


Reaction to code freeze – branch to avoid holding up development of version N+!.

Ideally trunk still needs to be stable prior to branching – no last minute high-risk changes that might be pulled.

Need integration branch when starting from a label.

Very few, carefully reviewed changes expected - only essential changes.

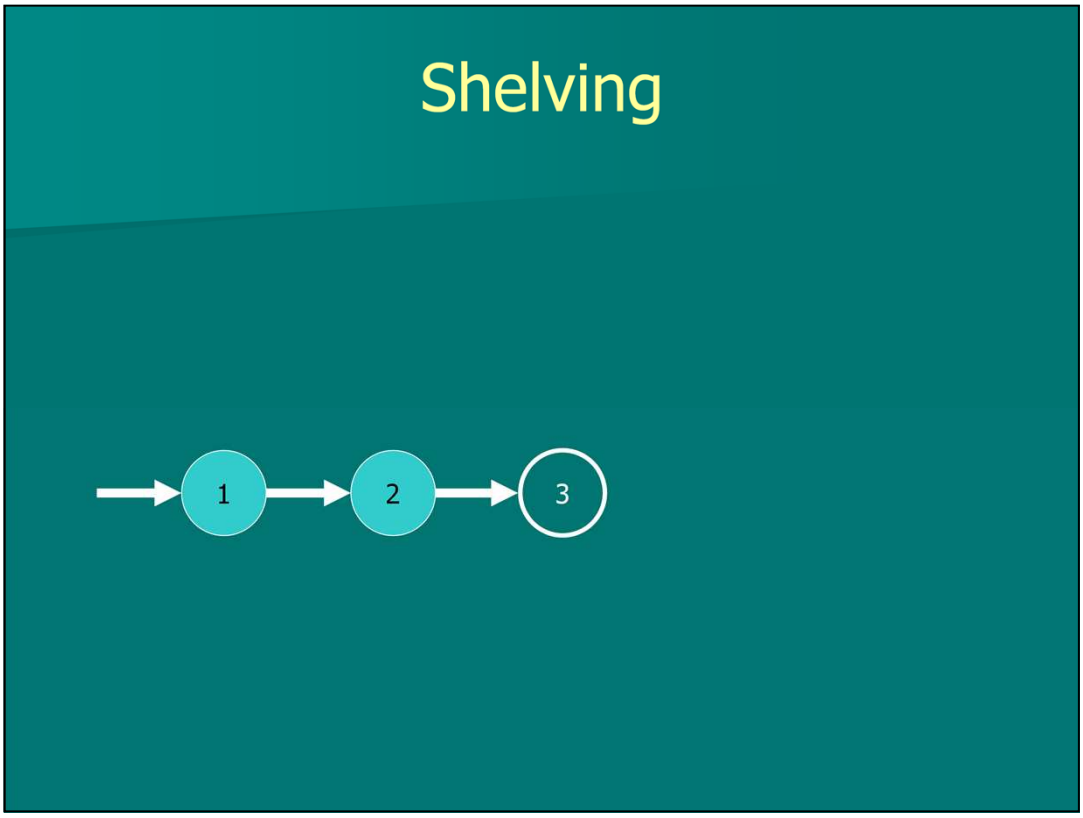


Branch for a specific feature (task) – often volatile in nature, e.g. a spike.

Or the developer may be volatile, e.g. new joiner.

Not necessary a single-developer branch, can allow multiple people to work more freely.

Easy to throw away with no residual effects.

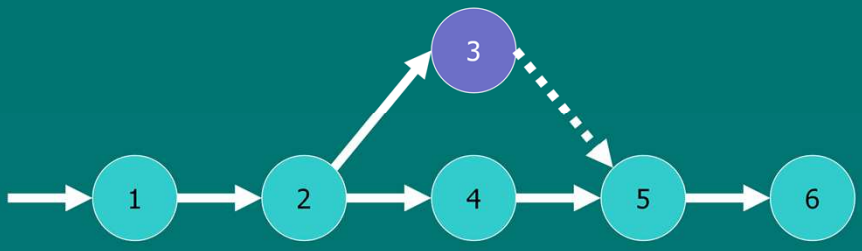


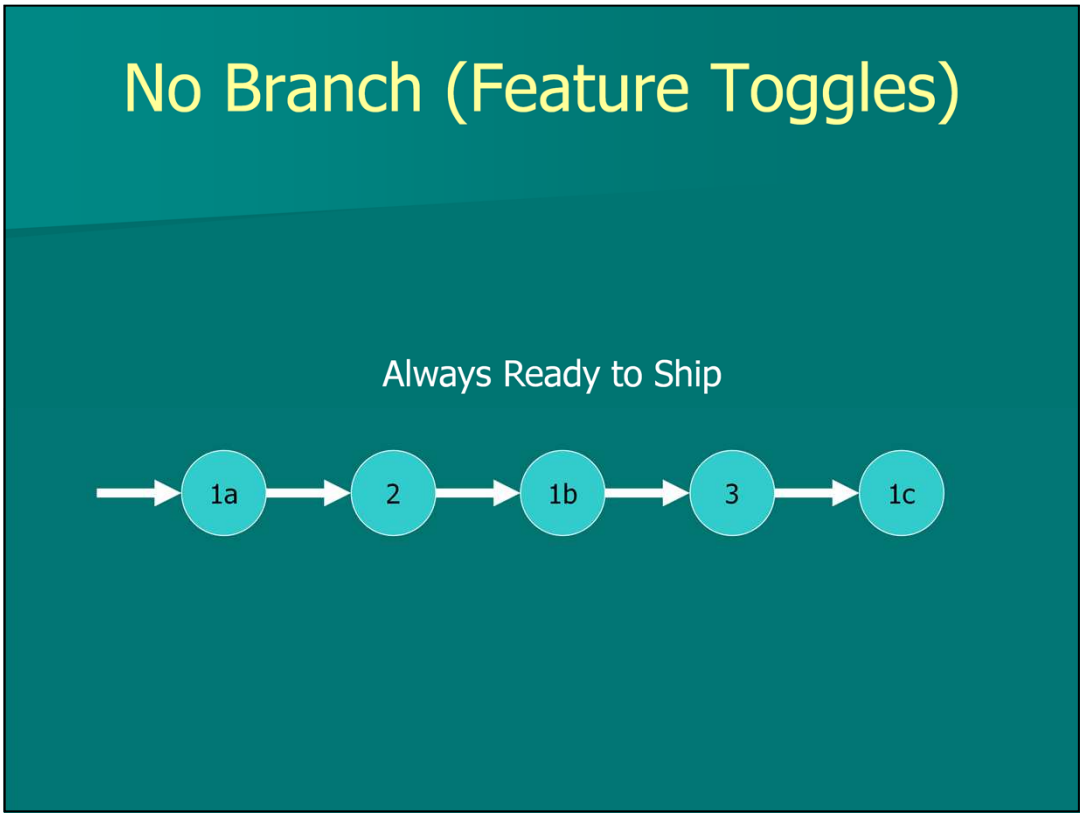
Very short-lived branch, effectively only one commit.

Put current changes to one side and integrate again later when dust has settled.

Supported natively by some, called stashing in Git, branch from working copy is an alternative in SVN.

Shelving





Break task down into much, much smaller tasks.

New code and refactorings don’t require toggling off, only changes (low risk, but not no risk)

Need to schedule clean-up after toggled on permanently.

Toggles can be compile-time (#ifdef) or runtime (.config entries).

Supports truly-continuous integration.

Merging

What's the collective non for a group of developers? A merge conflict.

Define "merge" as applying deltas to the content (merge/rebase applies to metadata).

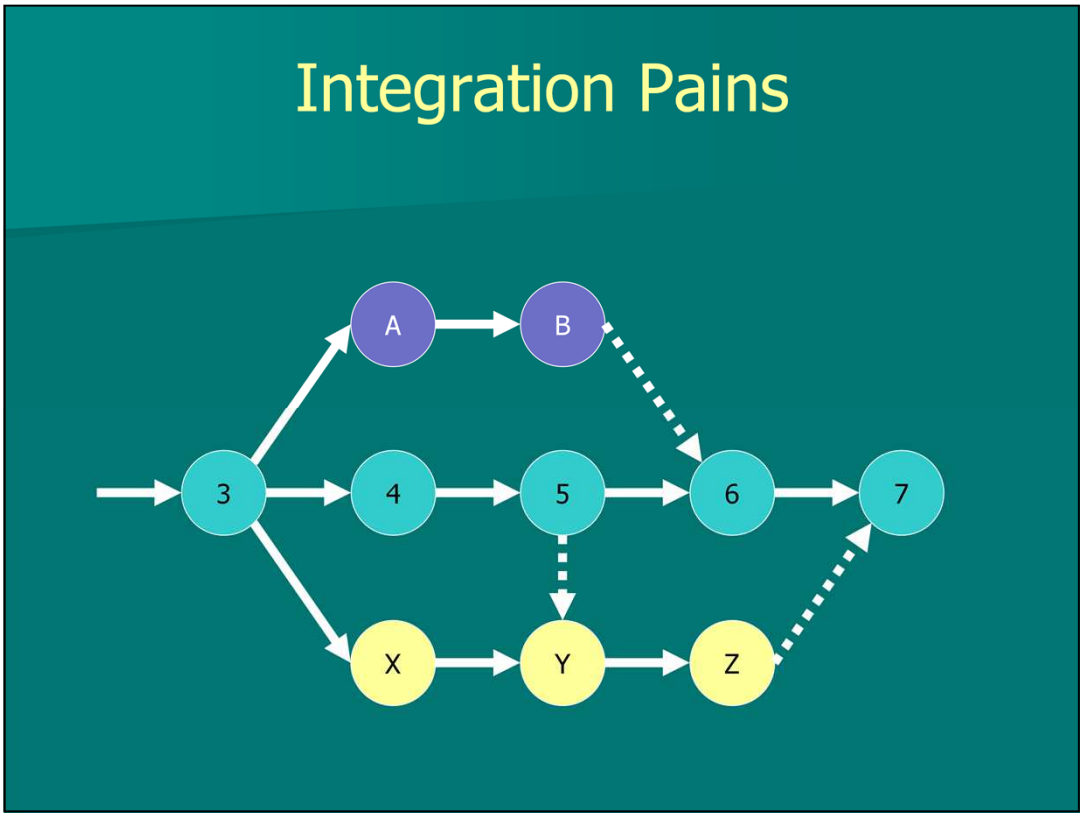
Avoid it, although every integrate is a mini-merge, but much more manageable.

Integrate often especially when not using integration branch directly.

Always merge from stable (release) -> volatile (development). Cherry picking is the reverse of this.

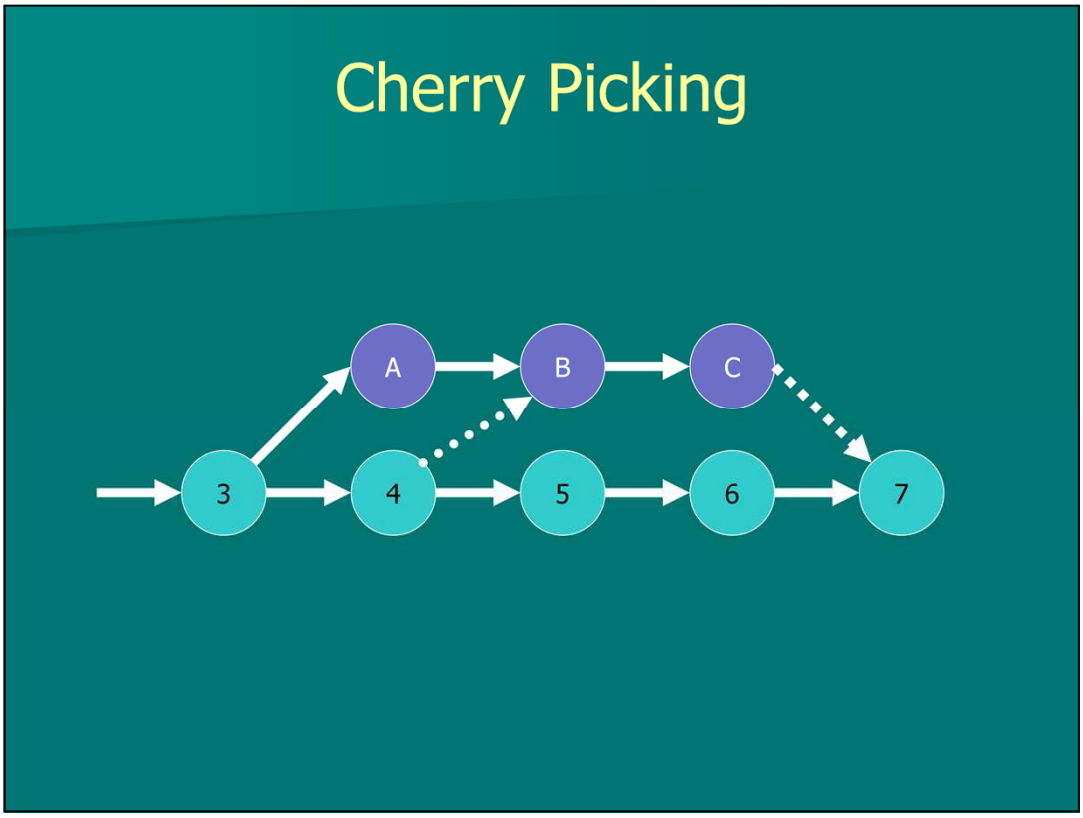
Attitude – start by considering "theirs" to be correct and "mine" to be wrong – Seagull Merge.

Tooling – two and three-way merging. Semantic merging. XML – merge as text often easier.



The release merge is easier due to small focused changes.

The feature branch merge can be harder because of the potential volume for change, e.g. refactoring.



Undesirable, but often a reaction to an overly long testing phase.

Small, focused commits make it easier to cherry pick as changes are isolated.

Heavy refactoring makes this much harder as the likelihood for dependent changes increases.

Changes can get lost on the merge back at the end.

Record a merge at the end as nothing has changed code-wise but the loop should be closed.

Committing

Define commit as “publishing”, so that’s commit + push in Git.

What to commit – source only (preferable), build artefacts (only as an optimisation, e.g. shared libraries).

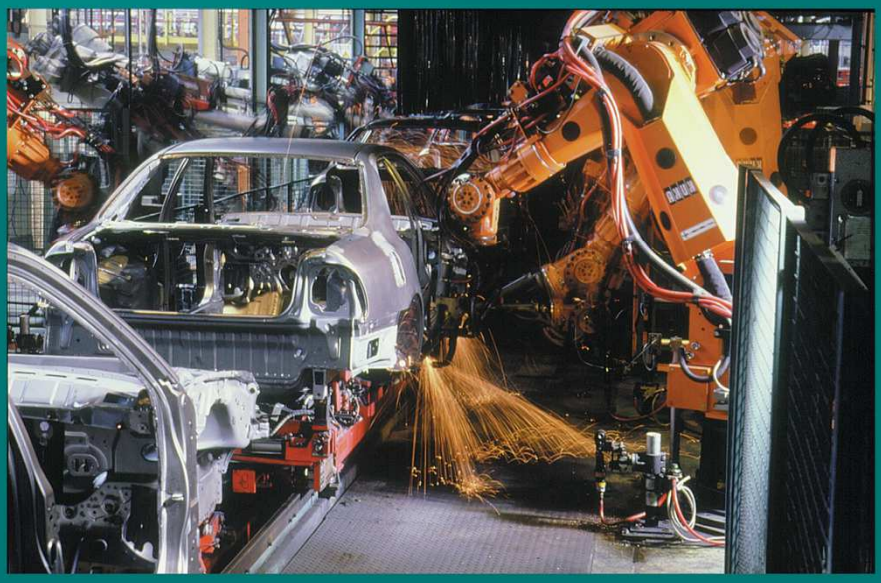
Diff all files to make sure the changes contains no mistakes and “reads” correctly.

Message should be short – the code says how the commit message says why.

Use of “and” is a smell that the commit should perhaps be split into multiple change-sets.

Provide a link to any supporting documentation, e.g. JIRA number – integration with other tools.

Building



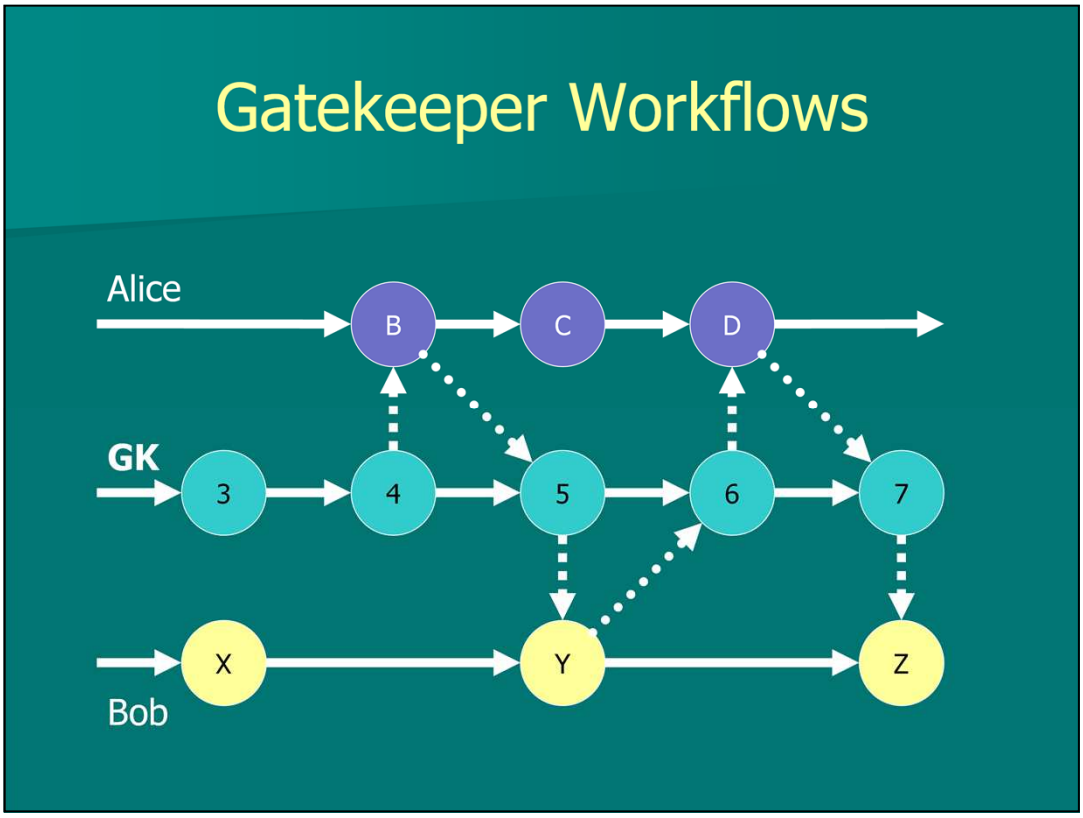
Commit should trigger the continuous integration server. Might need delay for non-atomic commits.

Optimistic workflow assumes commits are correct and should be ready to ship.

The only breakage should be environmental or long-running tests that can be elided by developers.

Build number should be auto-generated and baked into artefacts where possible.

Wipe workspace if you can afford it, else clean thoroughly to give same effect – no uncommitted hacks should taint the build.



Pessimistic workflow uses feature branches and build machine attempts to integrate.

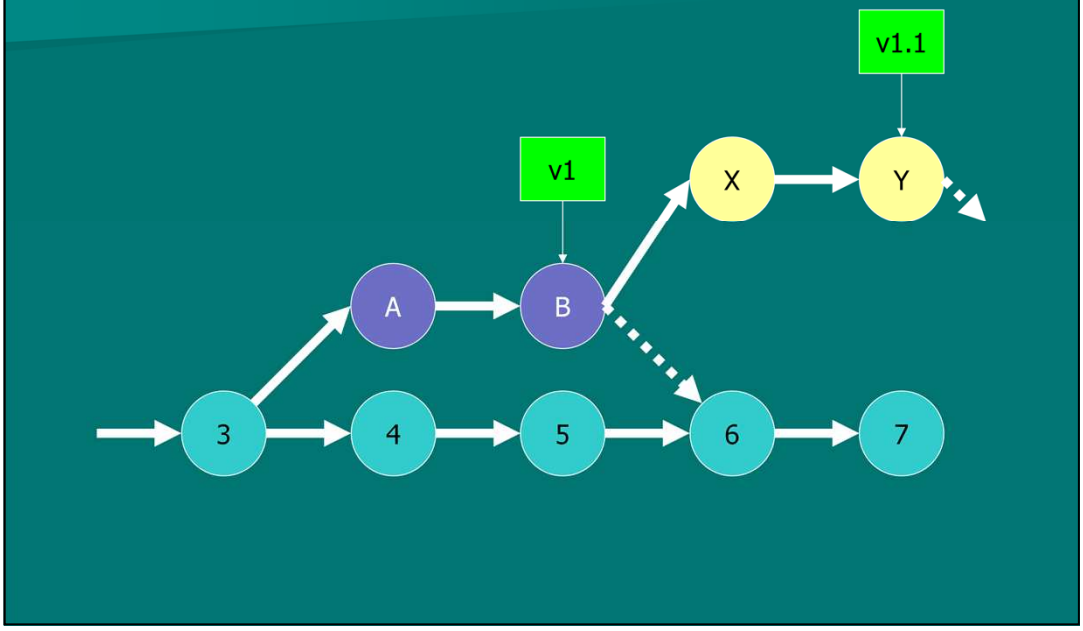
Labelling

Label to trace a deployed build back its constituent source parts.

Only need to label deployed builds as internal builds are not usually reported against.

More of an issue when you have multiple releases in the wild, e.g. beta testers, multiple live product versions, hotfixs.

Branching From a Label



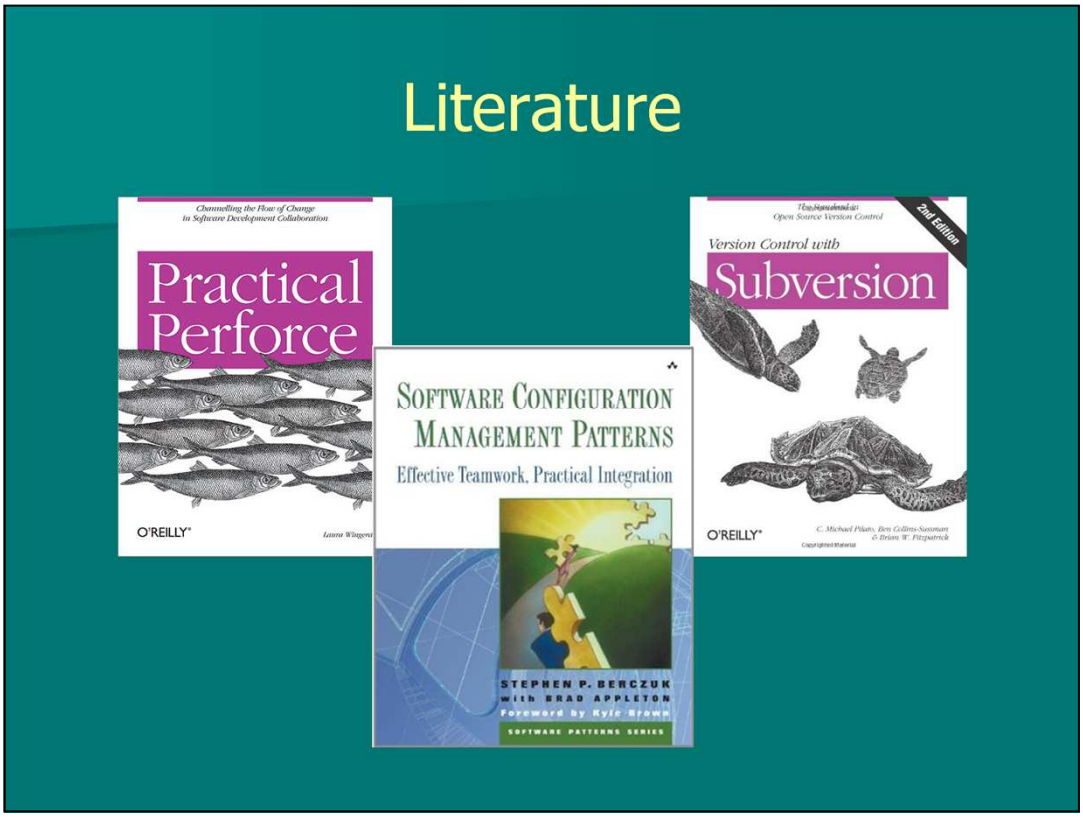
Archaeology

Tracing back through time to work out what or why something was done – the narrative.

Or perhaps mining trends within the codebase.

Blame – should be named “excavate”.

Investment needs to be put in at commit time to reward later in the day.



SCM Patterns book – old (doesn't even mention SVN!) but still useful.
Free chapter (7) from Practical Perforce about software evolution.
Martin Fowler's blog has posts about branches and toggles.

Questions?

No books, just a blog