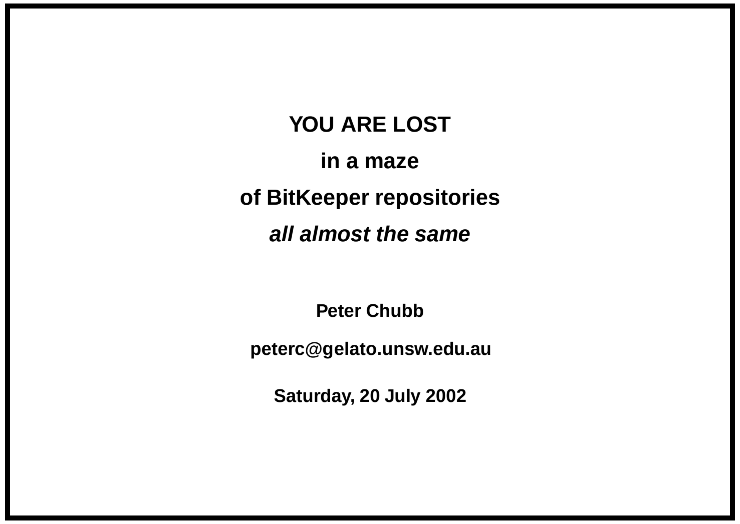
Slide 1



YOU ARE LOST
in a maze
# of BitKeeper repositories
all almost the same

Peter Chubb

peterc@gelato.unsw.edu.au

Saturday, 20 July 2002

A software change management system exists to manage changes in a body of software. Yes, I know that's tautology but...

What 'Management' means depends on the context. Some SCMs exist to support preexisting software process (e.g., ClearCase, Ingres 'Piccolo', Aegis); others were developed to provide the bare minimum without getting in the way of the developers (e.g., RCS, CVS, perforce).

All SCM systems keep logs of which developer changed each line of code and when, and have the ability to log what the change was for.

Some SCM systems can be put in a mode where all changes have to be allocated to a particular change request or bug report.

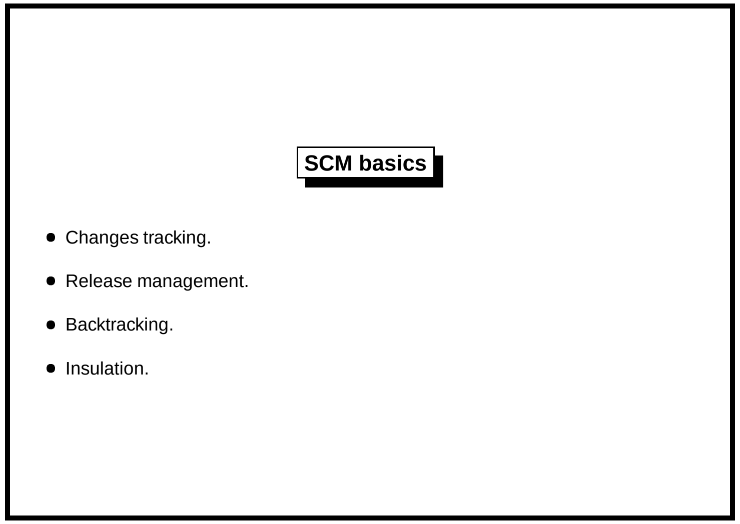
When a piece of software is finally released, it's important to be able to see exactly what's in it, especially as regards which bugs from the previous release are fixed; what features have been enhanced or added, etc. Much of this information is obtainable from the SCM logs. Moreover, the SCM logs can be used in some SCM systems to track the status of each file: whether it has been reviewed, tested, etc. So the SCM system works in with the software development engineering process.

When a bug report comes from a customer, one wishes to be able to reestablish the source state as at the release the customer has. Almost any SCM system will let you tag a particular set of files (or the entire repository) with a symbolic label, to allow recreation of the repository as of a particular time or state.

Also, if a particular set of changes causes a problem, one wishes to be able to back it out cleanly from the mainline (maybe transferring the change to a development branch).

When more than one developer is working on a software system, there's the problem of keeping them from treading on each others' toes. Ideally, each developer can work on his or her own copy of the source, and merge with a common version when things work.

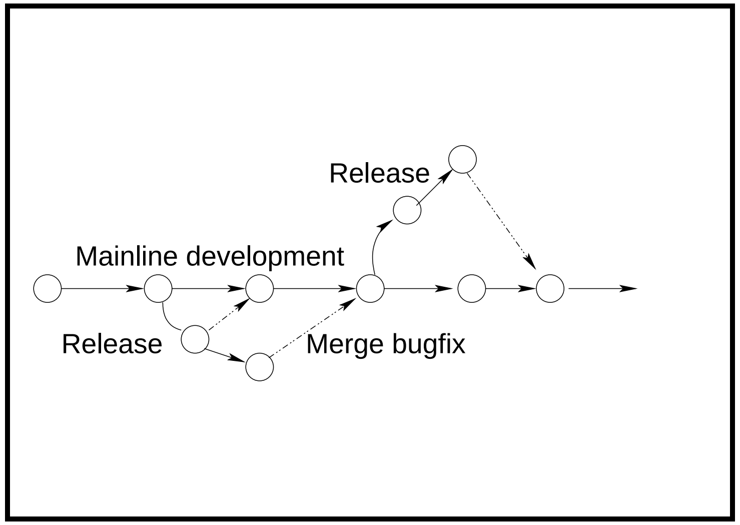
Slide 2



SCM basics

- Changes tracking.
- Release management.
- Backtracking.
- Insulation.

Slide 3

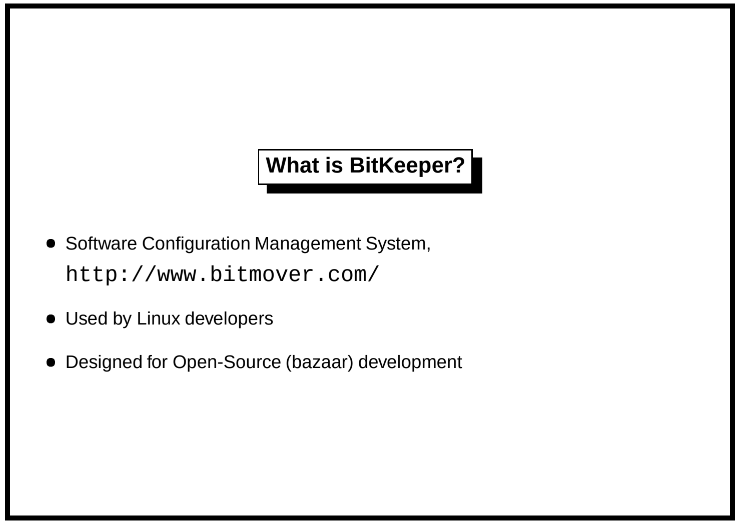


In a traditional software development process, there's a single main line of development, that progresses to some point, then is released. the release process usually involves branching a testing copy of the source, testing it rigorously, fixing bugs on that branch etc., while mainline development continues.

Branches are created for two reasons: to create a release (testing and bug-fixes happen on the branch), or to do experimental code that may or may not be merged into the mainline.

This doesn't match that well onto a 'bazaar' style development.

Slide 4



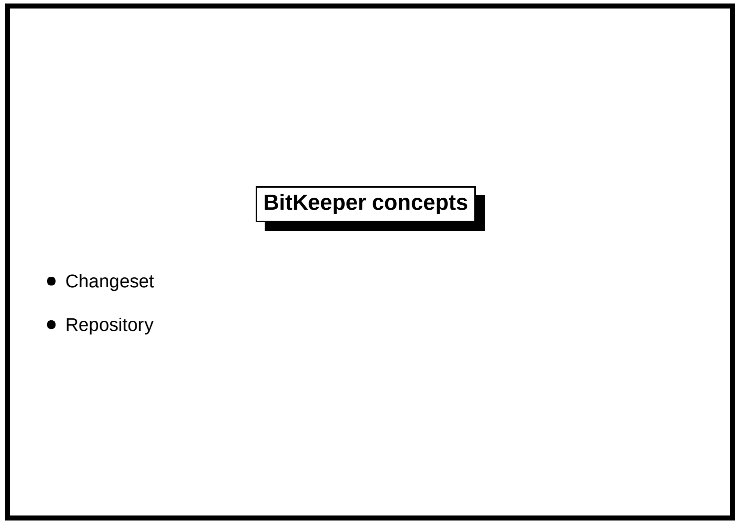
What is BitKeeper?

- Software Configuration Management System, `http://www.bitmover.com/`
- Used by Linux developers
- Designed for Open-Source (bazaar) development

BitKeeper is an SCM that's designed for bazaar-style open-source development. It's not perfect, but it works well enough.

Lots of Linux kernel developers are using BitKeeper, mainly because Linus is.

Slide 5



BitKeeper concepts

- Changeset
- Repository

A *changeset* is a collection of changes to files that have some commonality. For example, you fix a bug, that involves changing five files. The changes to those five files form a changeset.

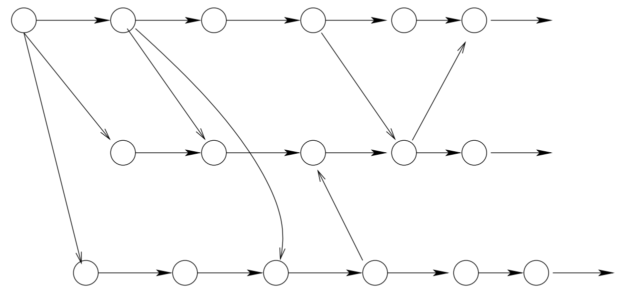
Changesets are immutable.

A *repository* is an ordered collection of changesets.

Each changeset has a comment, and can be given a tag.

Slide 6

BK model



Instead of a single main line of development, BitKeeper allows multiple parallel lines of development.

Now for the interesting bit: changesets can be transferred between repositories. BitKeeper can pull changesets from one repository into another, and then gives some assistance in merging overlapping changes (where the target repository has a change that touches the same file(s) as changes in the source repository).

Unfortunately at present you're limited to pulling *all* the changes from one repository into another, which limits the ways BitKeeper can be used (see below...)

Slide 7

Basic BitKeeper usage

- Creating a repository
  - import (from plain source, or from CVS, SCCS or RCS)
  - clone (from another repository)
  - Create empty repository `bk setup`

You have to start with something... You can either create an empty repository with `bk setup` or import code from somewhere else. Most Linux kernel hackers will do

```
$ bk clone bk://linux.bkbits.net:8080/linux-2.5 linux-2.5
$ cd linux-2.5
$ bk -Ur co
```

to get a usable kernel source tree; then happy hacking.

Slide 8

Making changes

- Same as SCCS: bk edit; bk delta
- Or can use RCS-like commands: bk co -l; bk ci -u

Whilst editing files, BitKeeper feels like SCCS. And yes, emacs vc mode does work.

Slide 9

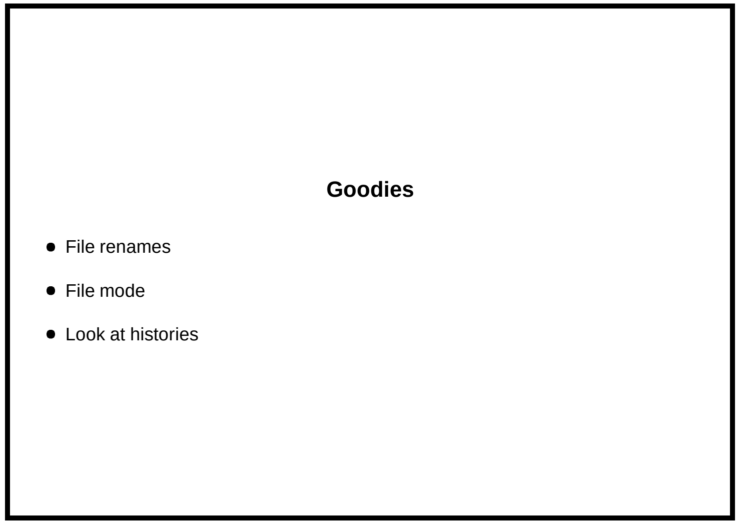
More on Changesets

- Groups of deltas with some commonality of purpose
- Repository a sequence of changesets
- Deltas committed to a changeset, thence immutable.

You can do the standard SCCS-like things as much as you like (`bk edit` *filename*, `bk delta` *filename*, `bk fix` *filename*), then use either `bk commit` to commit the changes you've made to a changeset, or if you're running X, use the GUI tool `bk citool`, which lets you choose easily which changes belong to the changeset.

Once a changeset has been committed it's immutable.

Slide 10



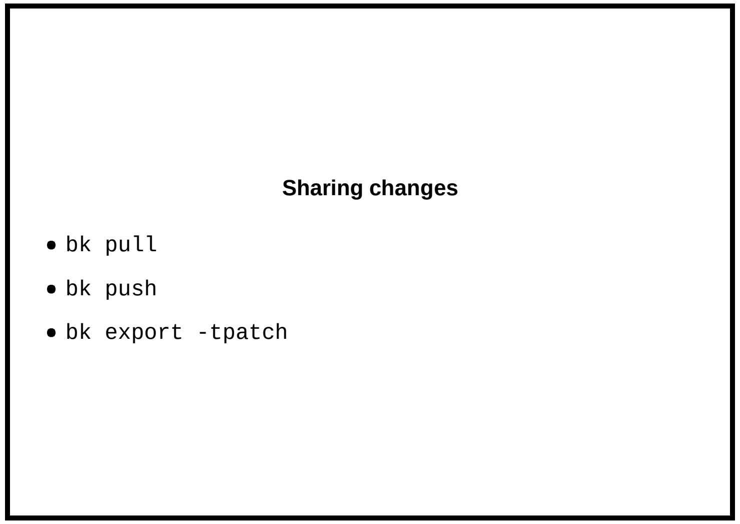
Goodies

- File renames
- File mode
- Look at histories

Unlike most other SCM systems, BitKeeper handles renames and chmods and symbolic links. You can do `bk mv`*source destination*, and then commit that change to a changeset at your leisure. Likewise, `bk cp` *source destination* and `bk chmod`*mode file* work the way you'd expect.

And bitKeeper has a very good set of GUI tools for looking at change history.

Slide 11



Sharing changes

- `bk pull`
- `bk push`
- `bk export -tpatch`

When there are multiple repositories (and if there aren't to start with, there soon will be), you can import the changes from one repository into another, using `bk pull` *from* or `bk push` *to*. These commands transfer all changesets in the source into the destination, leaving the destination a superset of the source.

Slide 12

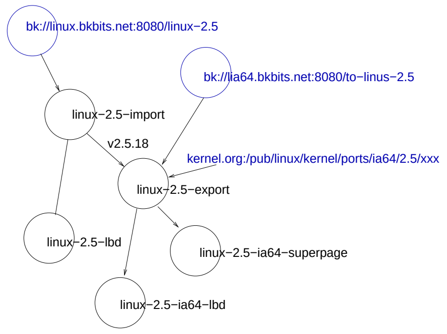
Problems

- Often need extra scaffolding for testing
- Can't easily isolate changes

OK, so you have some change to the Linux kernel you want to test on various platforms to make sure that it works. Only problem is that the current head-of-tree from Linus is broken, and won't run on any of your test platforms. You want a clean version with just the tested change in to export to Linus...

Slide 13

My use of BitKeeper


bk://linux.bkbits.net:8080/linux−2.5
bk://ia64.bkbits.net:8080/to−linus−2.5
linux−2.5−import
v2.5.18
kernel.org:/pub/linux/kernel/ports/ia64/2.5/xxx
linux−2.5−export
linux−2.5−lbd
linux−2.5−ia64−superpage
linux−2.5−ia64−lbd

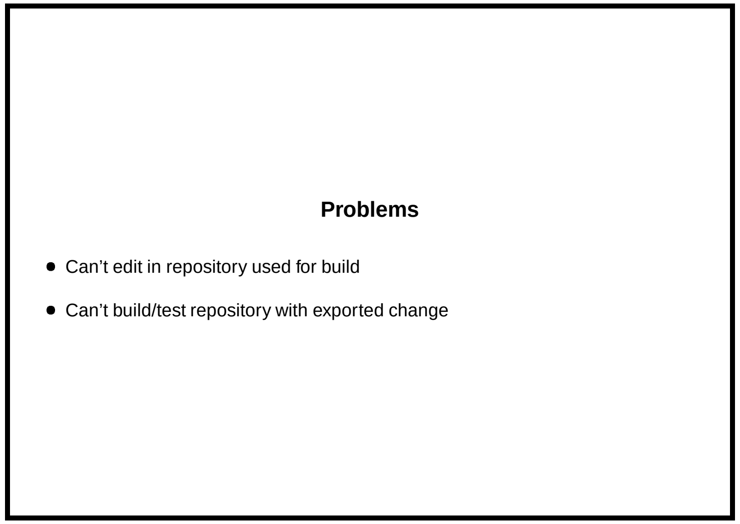
So what you do is branch. Branching in BitKeeper happens by cloning a repository. (If two related repositories are on the same filesystem, BitKeeper can use hard links to reduce disc space).

Here the two repositories in blue are on some external machine. They're cloned locally to reduce network traffic.

To run Linux on IA64, you need extra patches from kernel.org. Some of these changes are in the to-linus source tree, others are not. In addition, there are some changes in the to-linus tree that haven't yet made it into the kernel.org patch. I integrate these into linux-2.5-EXPORT.

Now, the changes for supporting large block devices (that I'm trying to test) are done first in a clone of linux-2.5-import (which is backtracked to version 2.5.18), and tested there on Pentium; then pulled into the IA64 tree and tested there; and pulled into a proper clone of the 2.5.26 (at present) tree and tested there. The latter is the one that's used to generate a patch to send to LKML, and is exported for Linus.

Slide 14



Problems

- Can’t edit in repository used for build
- Can’t build/test repository with exported change

Note, this means that all substantive changes have to be made in the 2.5.18 tree, and then pulled into the other trees for testing. And the tree that is exported is never the tree that’s tested. Moreover, you can’t actually do this — BitKeepper won’t let you pull from a tree that has changes from a common ancestor that you don’t have. So you’re stuck with exporting and importing patches.

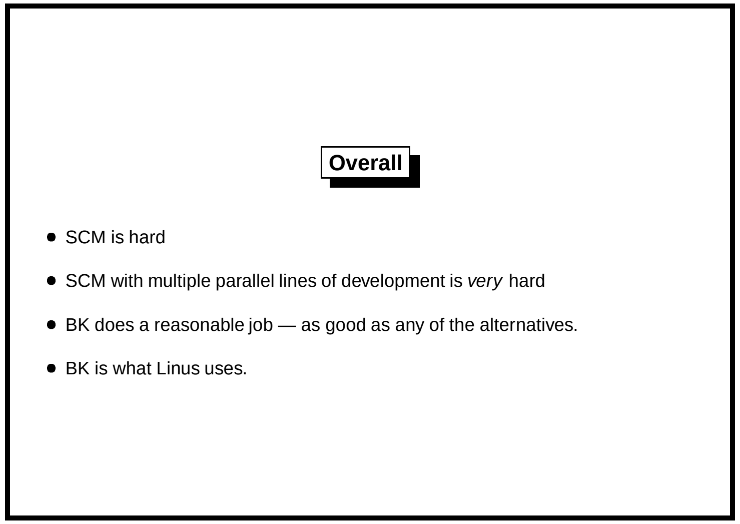
On this path lies danger.

Moreover, it’s very easy to make a minor fix in one repository, e.g., to fix compilation or remove a warning, while one’s working/building/testing in that repository. It’s hard to remember to then replicate the change into the mainline.

Moreover, it’s easy to create a new repository to try out an idea, make some fixes in it, then delete the repository, having forgotten about the fixes that are in there (especially when you’re working part time on the problems).

There are some changes to BitKeeper in the pipeline that’ll allow a single changeset to be pushed into another set of trees.

Slide 15

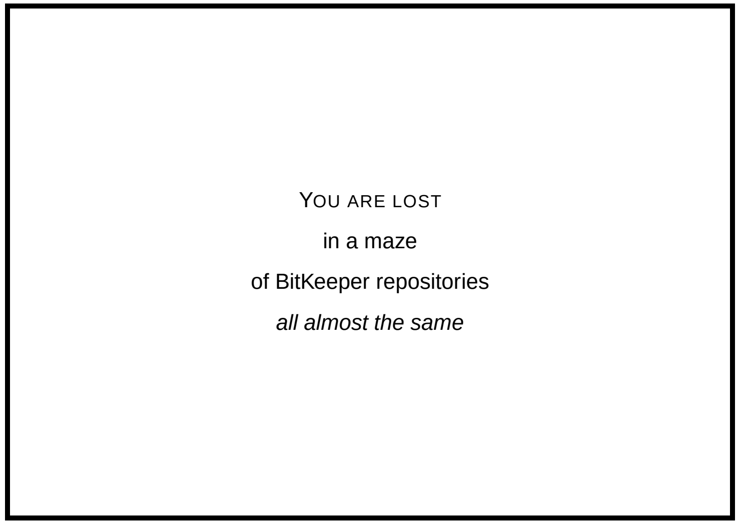


Overall

- SCM is hard
- SCM with multiple parallel lines of development is *very* hard
- BK does a reasonable job — as good as any of the alternatives.
- BK is what Linus uses.

So that’s what I’m using too.

Slide 16



YOU ARE LOST

in a maze

of BitKeeper repositories

all almost the same