



The Bazaar Version Control System

Michael Hudson, Canonical Ltd
michael.hudson@canonical.com



What is Bazaar?

- Bazaar is a Distributed Version Control System (DVCS)
- You probably know what a VCS is by now: CVS, Subversion, Perforce...
- “Distributed” means there is no privileged central location
- Bazaar aims to be **safe, friendly, free** and **fast**



What is Bazaar?

- Bazaar is a Distributed Version Control System (DVCS)
- You probably know what a VCS is by now: CVS, Subversion, Perforce...
- “Distributed” means there is no privileged central location (technically speaking)
- Bazaar aims to be **safe, friendly, free** and **fast**



Getting Started With Bazaar

- “`bzr init`” – Creates an empty branch in the current directory
- “`bzr add`” – Schedules the addition of all the files in the current directory to the branch
- “`bzr ci -m 'initial import'`” – Commits these files to the branch



Working With Others

- “`bzr push sftp://me.example.net/proj/branch`” – Uploads or updates branch data on the server (no server software required!)
- “`bzr branch http://you.example.net/proj/branch`” – Get a copy of a remote branch (no server software here either)



Working With Others

- “bzd pull” – Update your copy from the server (if no changes)
- “bzd merge” – Combine remote changes with local commits



Some Concepts

- Working Tree – collection of version controlled files and directories
- Revision – A snapshot of a working tree and the fundamental object in Bazaar
- Branch – An ordered series of revisions
- Repository – A place where revisions are stored



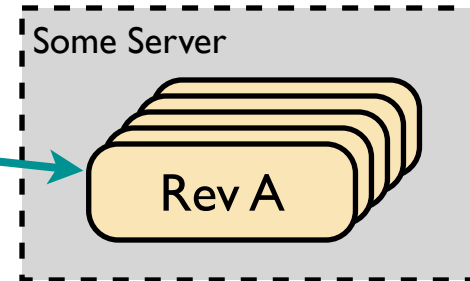
Some More Concepts

- Commit – the action of making a new snapshot of a tree, i.e. creating a new revision
- Delta – the difference between two revisions
- Merge – determining the outstanding revisions in one branch and applying them to another



Some Pictures :-)

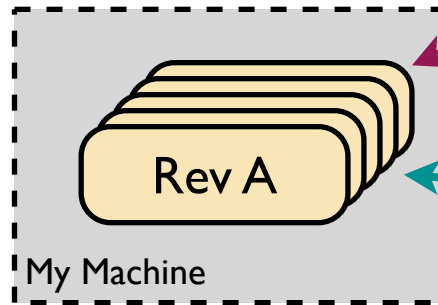
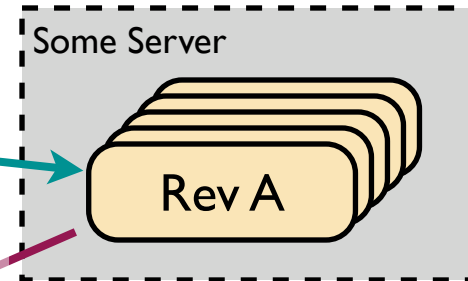
The mainline
branch for your project,
living on a server





Some Pictures :-)

The mainline
branch for your project,
living on a server

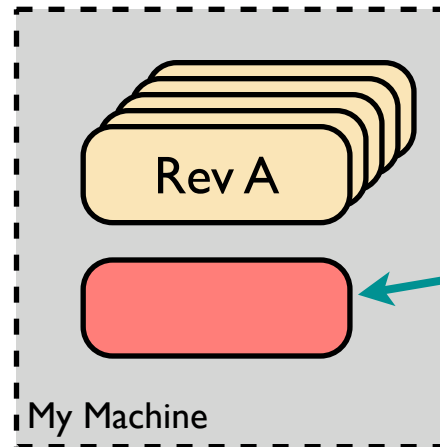
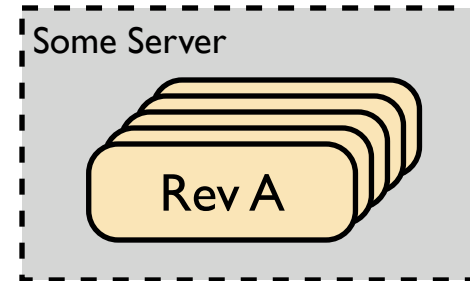


branch

You make a new **branch** on
your own machine to
develop some feature



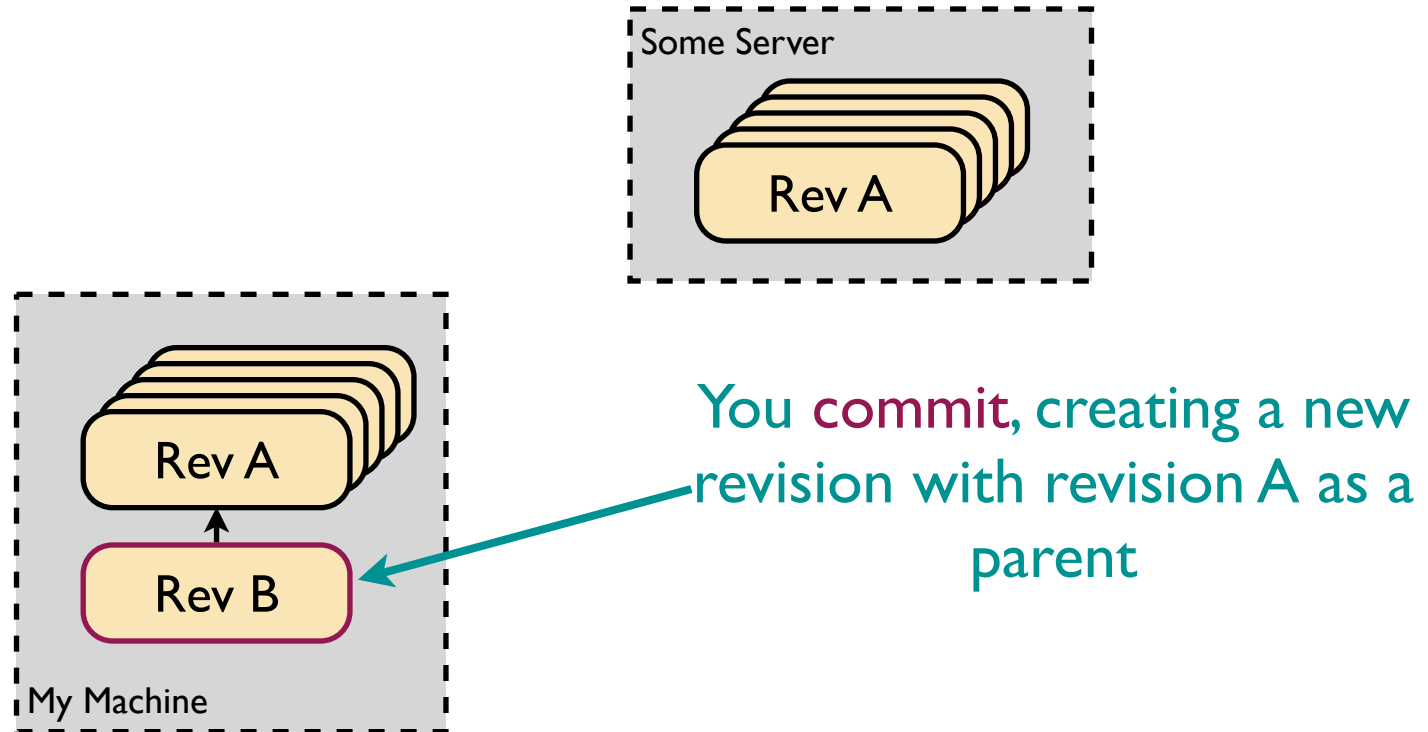
Some Pictures :-)



You make a start to your changes on your machine

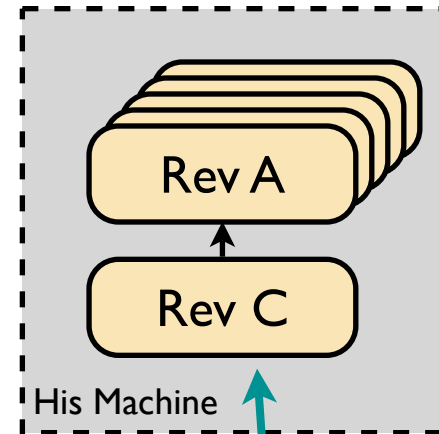
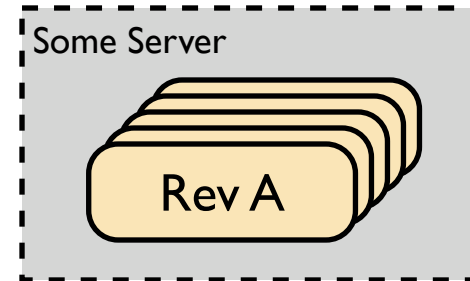
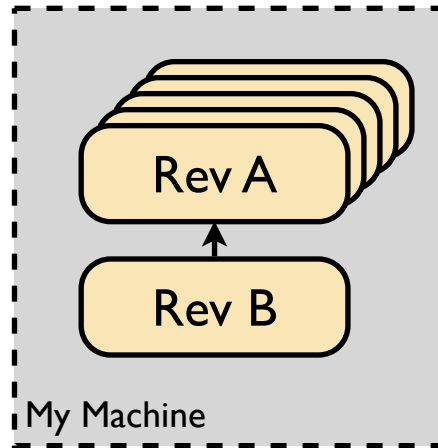


Some Pictures :-)





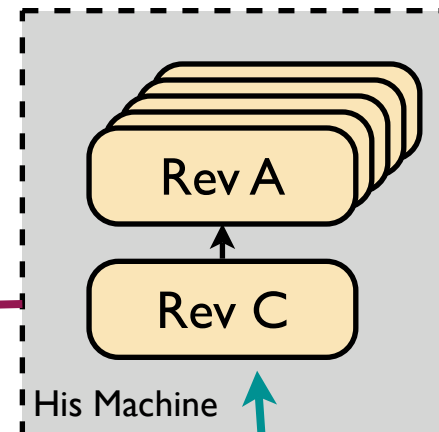
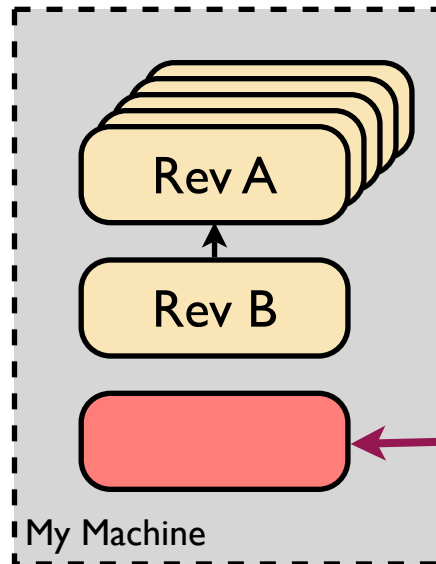
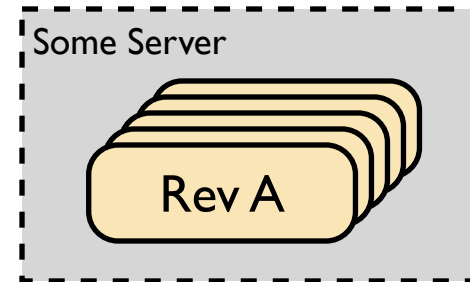
Some Pictures :-)



You hear that your
colleague has done some
work that will help you



Some Pictures :-)



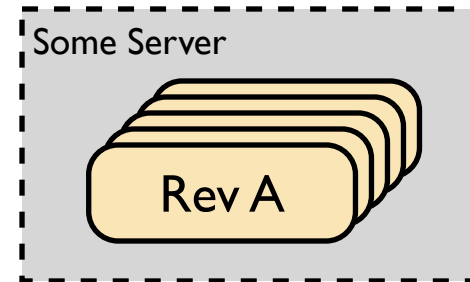
merge

So you merge his
changes into your
branch

You hear that your
colleague has done some
work that will help you

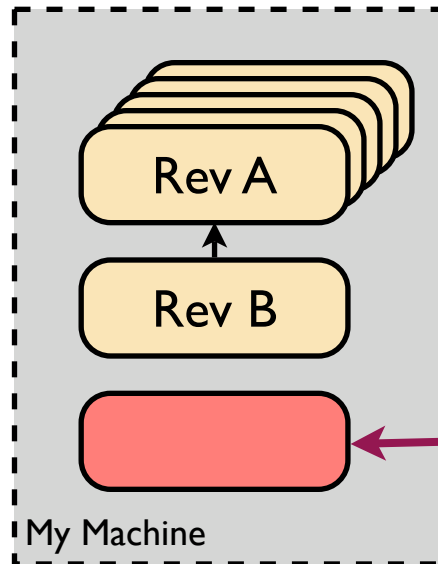


Some Pictures :-)



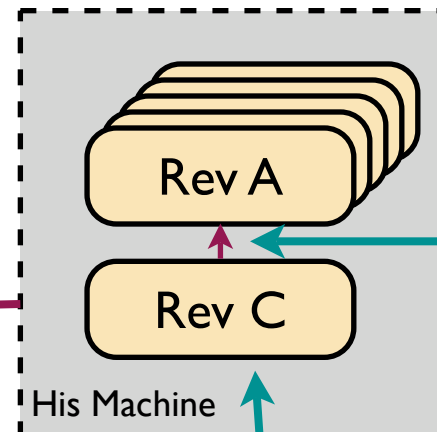
Bazaar

determines that
it's only the delta
between
revisions A and
C that need to
be applied to
your tree



So you merge his
changes into your
branch

merge

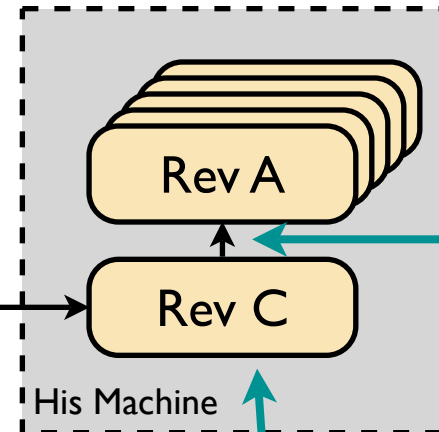
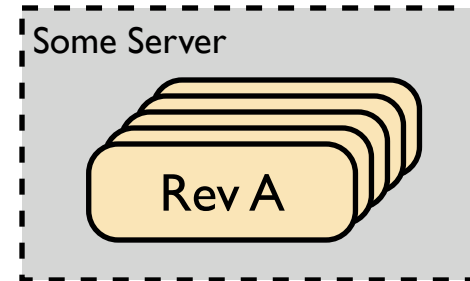
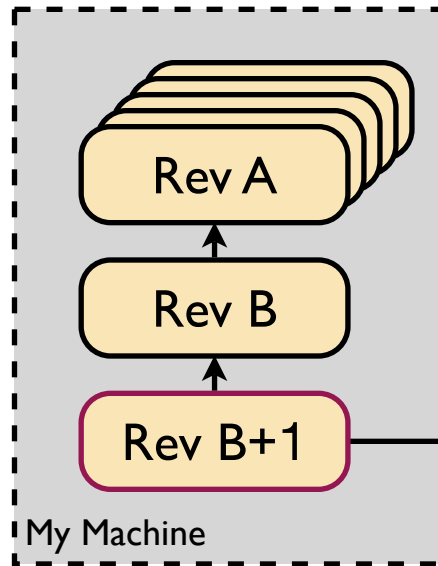


You hear that your
colleague has done some
work that will help you



Some Pictures :-)

So you merge his changes into your branch then **commit**, creating another new revision, which has two parents

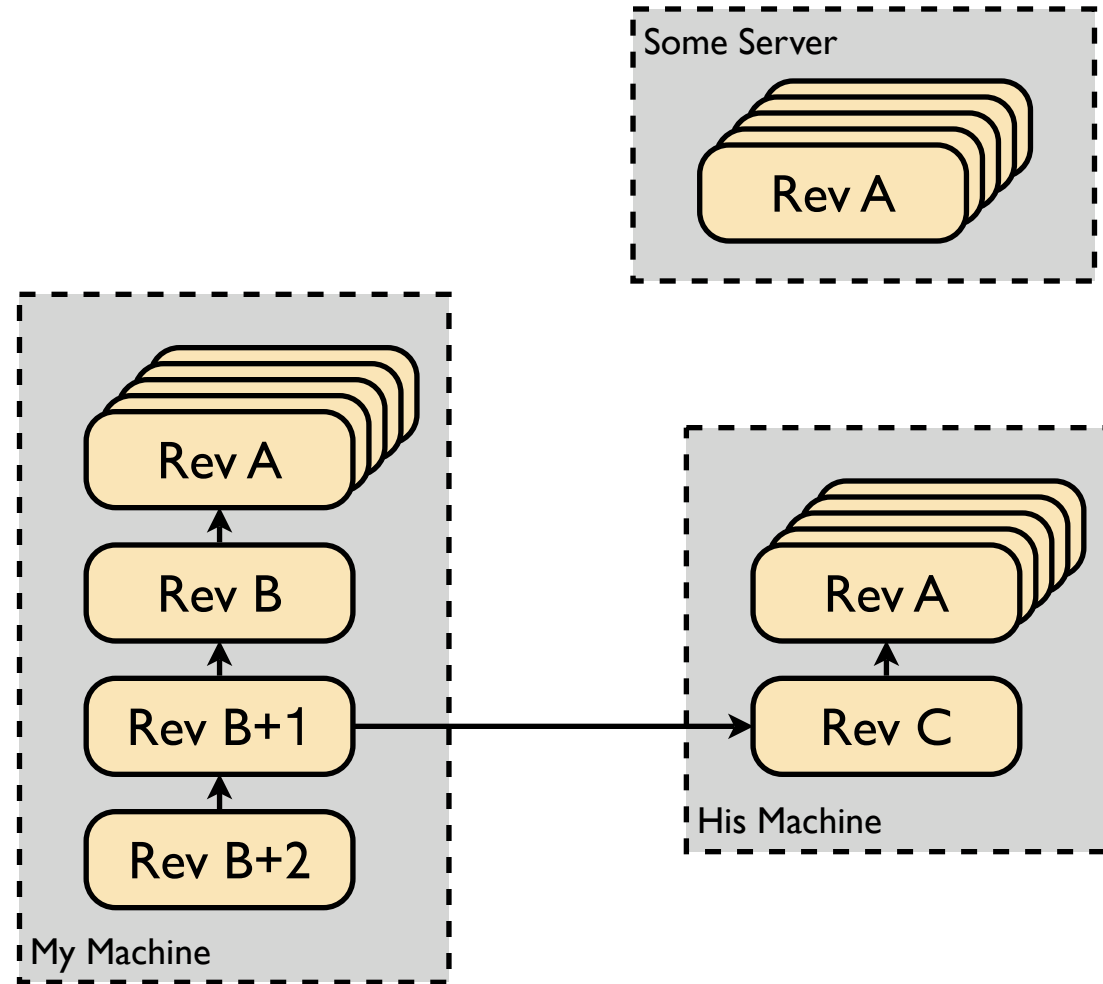


Bazaar determines that it's only the delta between revisions A and C that need to be applied to your tree

You hear that your colleague has done some work that will help you



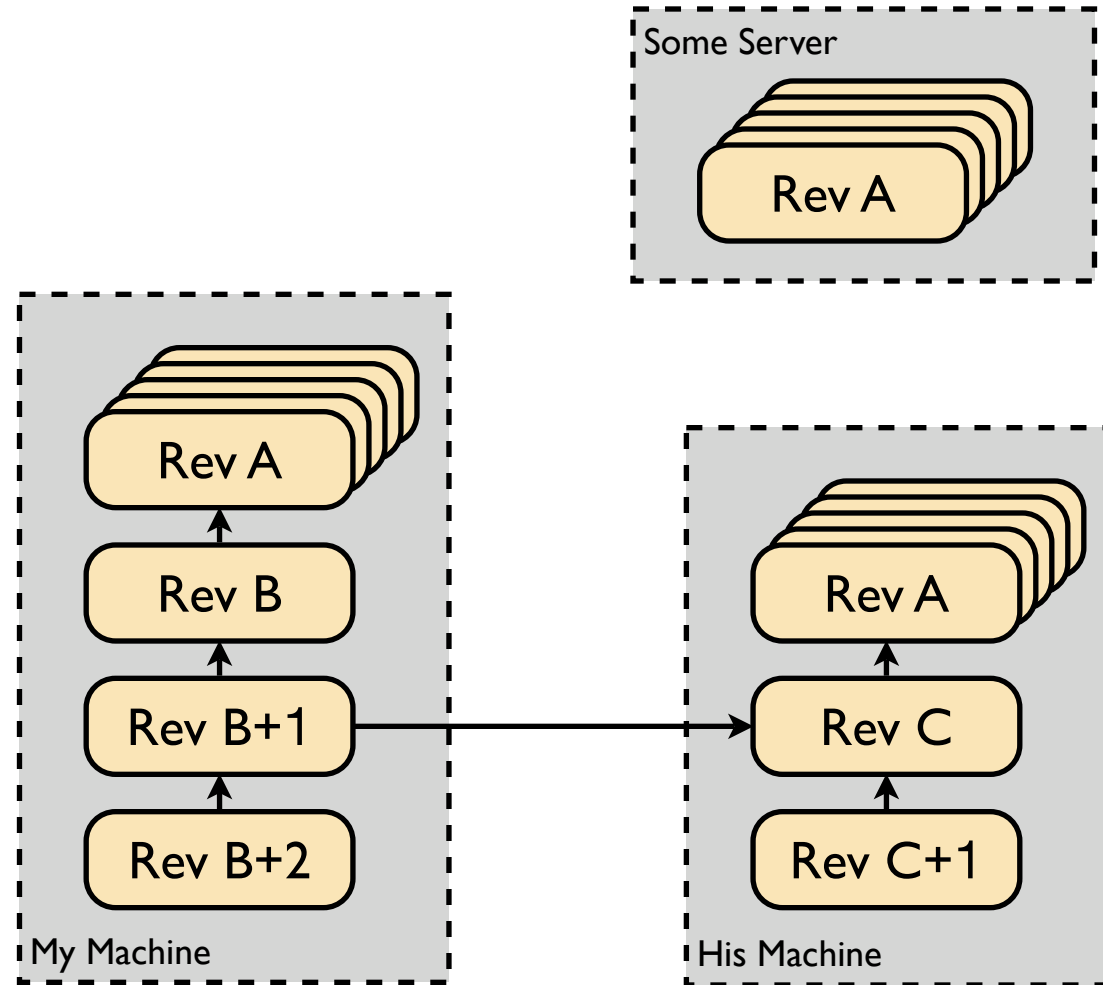
Some Pictures :-)



You go on with your work,
committing as you go



Some Pictures :-)

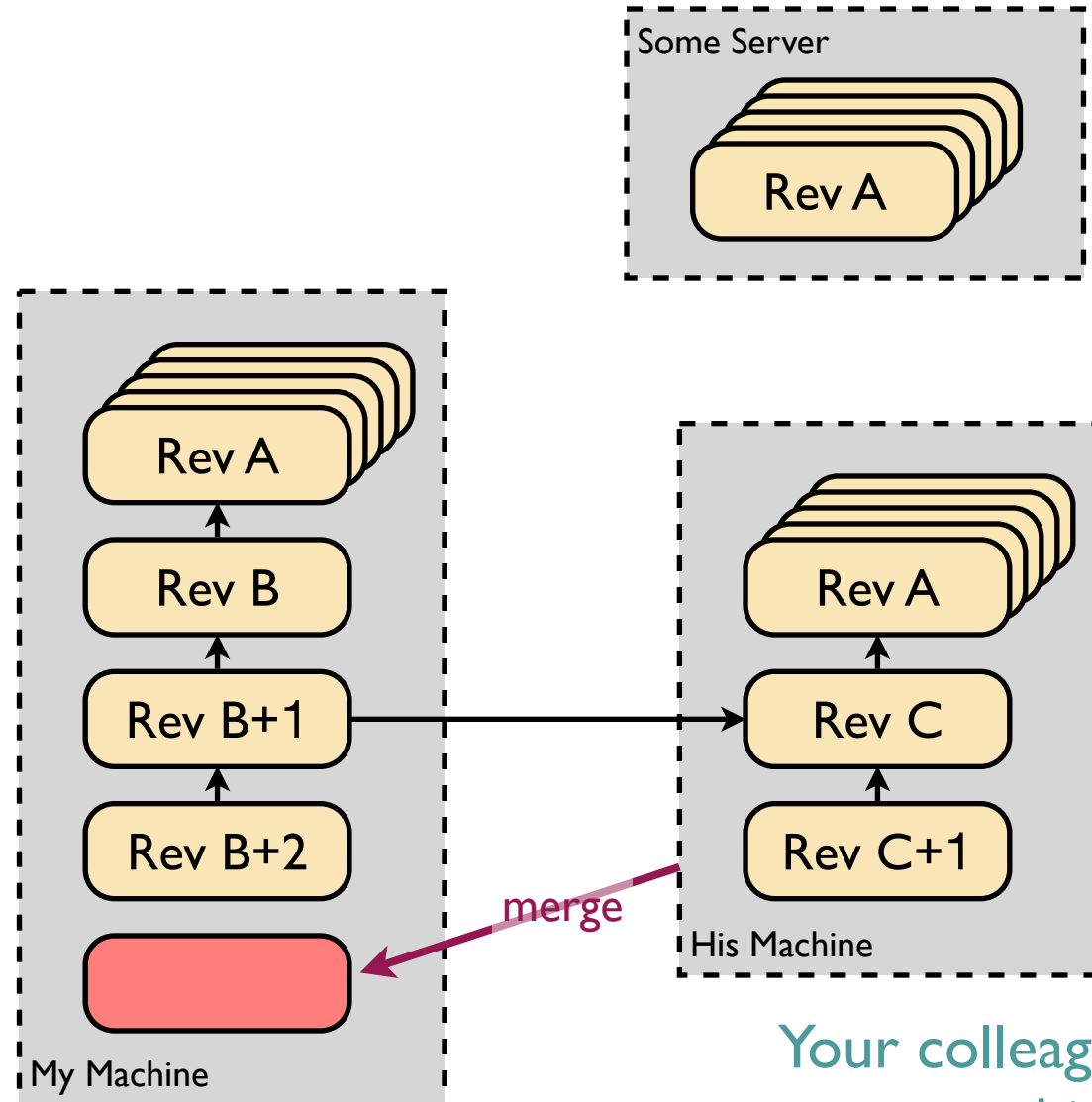


Your colleague fixes a bug in
his code



Some Pictures :-)

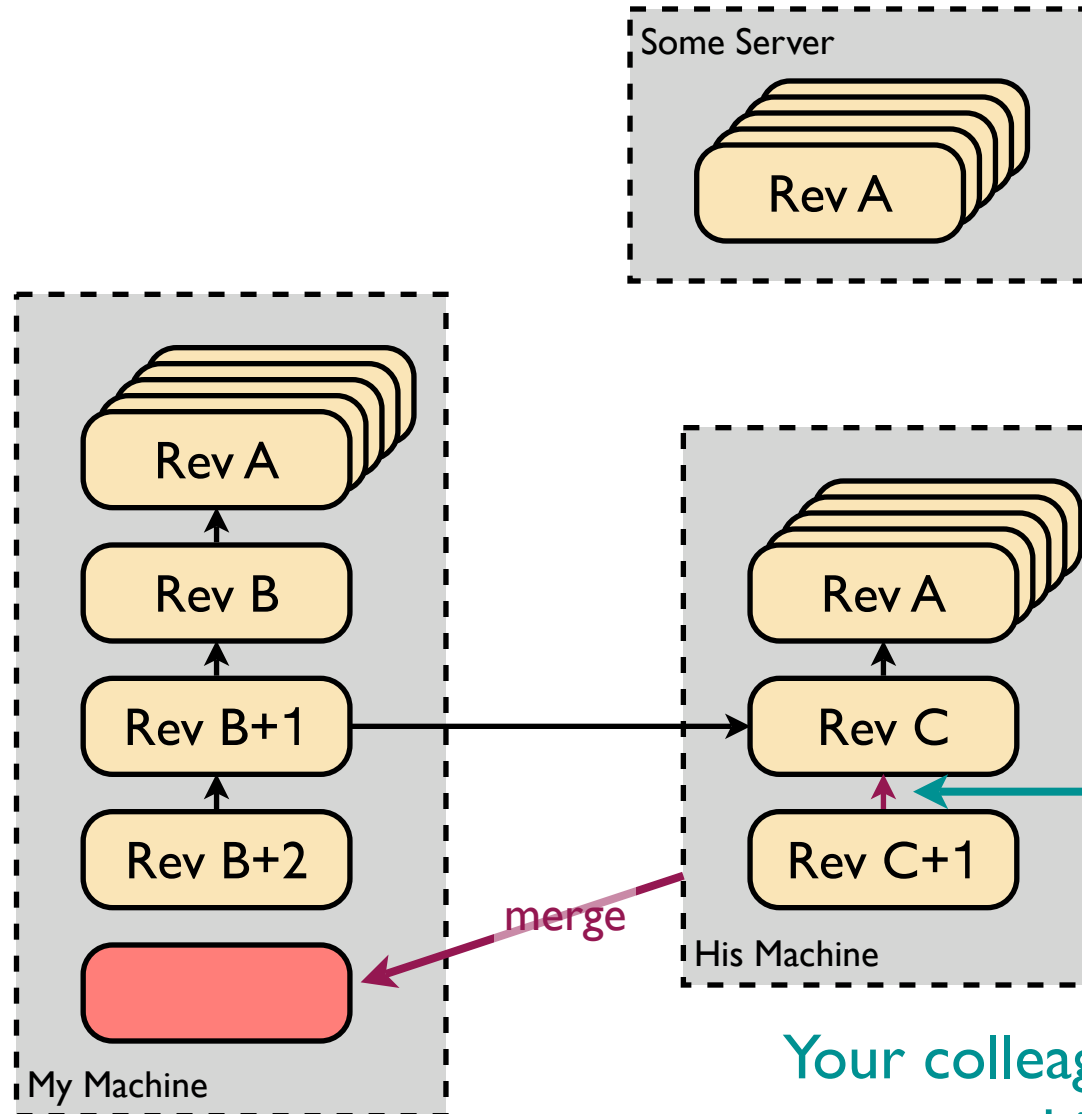
So you merge
again from him



Your colleague fixes a bug in
his code



Some Pictures :-)



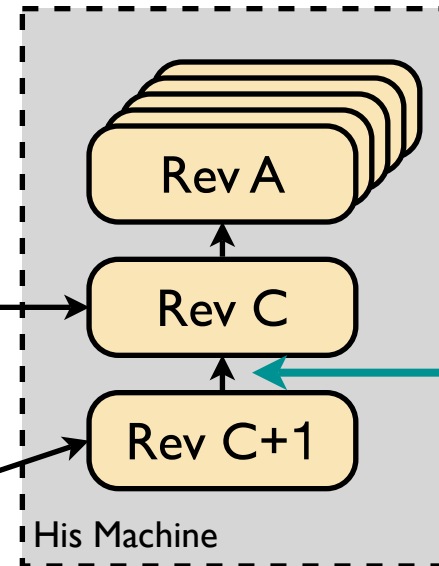
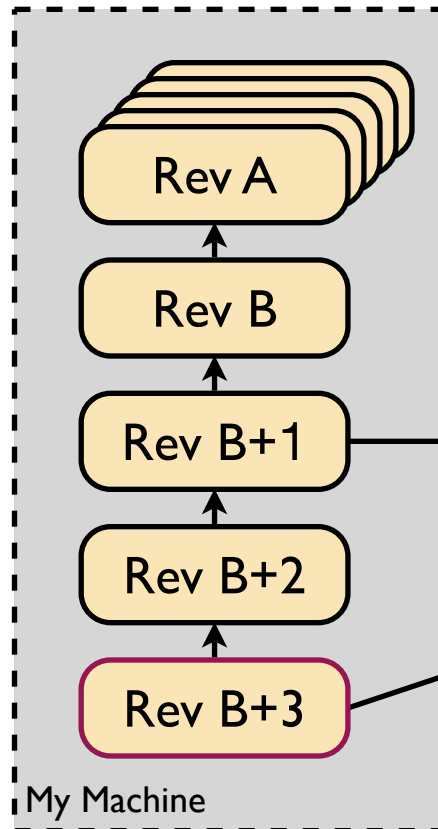
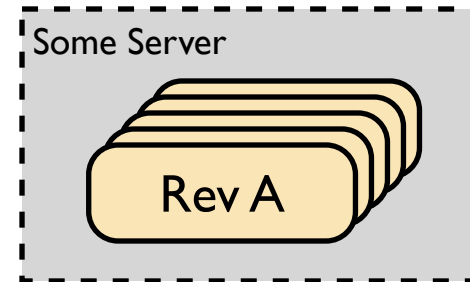
So you merge
again from him

Bazaar knows you
already have
revision C, so it's
only the delta
between C and
C+1 that needs
to be applied to
your tree

Your colleague fixes a bug in
his code



Some Pictures :-)



So you merge
again from him

Finally, you
commit again to
create a revision
containing the fix

Bazaar knows you
already have
revision C, so it's
only the delta
between C and
C+1 that needs
to be applied to
your tree

Your colleague fixes a bug in
his code



Workflows

- Bazaar is extremely flexible – confusingly flexible, perhaps – when it comes to workflows
- Two common workflows:
 - Centralized – all data lives on a single server, like Subversion
 - Distributed – data spread around several machines, no intrinsically privileged location



A Centralized Workflow

- The most Subversion- or CVS-like workflow
- Put a branch on a server
- Each developer runs

```
bzr co sftp://bzr.example.com/proj/trunk
```

(or possibly “`co --lightweight`”)
- Then “`bzr diff`”, “`bzr ci`”, “`bzr st`” work much like Subversion



Mainline Branches

- The workflow on the previous slide has some problems:
- Everyone is working in the same branch, potentially interfering
- Not much gain over Subversion, really
- Better to have a *mainline* branch that releases and/or deployments are made from



Mainline Branches

- The mainline, or trunk, branch reflects the currently blessed version of the code
- As such, changes should only go in when they are ready
- So each should be developed in a branch, separate from mainline



Protecting Mainline

- It's an obvious, and good, policy to say “the tests must always pass on mainline”
- The Patch Queue Manager (PQM) can enforce this (<http://launchpad.net/pqm>)
- An email-driven robot that guards a branch
- Listens for “merge requests” and runs the test suite before the change is applied to the branch



Better Centralization

- Create a new branch on the server:

```
bzr branch \  
    sftp://bzd.example.com/proj/trunk \  
    sftp://bzd.example.com/proj/feature-X
```

- Get the branch:

```
bzr co [--lightweight] \  
    sftp://bzd.example.com/proj/feature-X
```

(the “cbranch” command from the bzrtools plugin reduces the amount of typing here)



Better Centralization

- Make changes

```
cd feature-X  
vi src/whatever/blah.c  
bzd ci -m "fantastic changes"
```

- When done, merge your changes:

```
cd ../trunk  
bzd merge ../feature-X  
bzd ci -m "merge feature-X"
```



Working Distributedly

- A distributed workflow has no intrinsically preferred location (e.g. the server in the above examples)
- Developers maintain branches in their own areas



Working Distributedly

- Get your own branch:

```
bzr branch \  
http://bzr.example.com/proj/trunk \  
~/src/proj/feature-X
```

- Make changes:

```
cd ~/src/proj/feature-X  
emacs src/whatever/blah.c  
bzr ci -m "fantastic changes"
```

(difference here is that changes stay local)



Working Distributedly

- Share your code:

```
bzr push \  
    sftp://my.example.net/proj/feature-X
```

- Get another's code:

```
bzr merge \  
    http://example.org/~bob/feature-Y  
bzr ci -m "merge in Bob's changes"
```



Working Distributedly

- When you're done (maybe after a code review, etc), you want to get your changes into mainline
- If you let developers make commits on mainline, then it's as for the centralized workflow
- With PQM (and the pqm-submit plugin), it's
“`bzr pqm-submit -m "merge feature-X"`”



Pros and Cons

- The main advantage of a centralized workflow is that it's easier to find out what people are doing and get their code
- The advantage of distributed development is that it is more flexible (you can work on a train and still make commits) and requires less granting of permissions



Mixing and Matching

- The two approaches quickly sketched above are only two points on a continuum of processes, there are other possibilities
- Some people can work in a centralized fashion, some distributed
- “`bzr ci --local`” can be used to commit to a checkout without accessing the server (e.g. on a flight)



Mixing and Matching

- In fact, the only difference between “`bzr co <URL>`” and “`bzr branch <URL>`” is that the former creates a “bound branch”
- A “bound branch” is one where commits are duplicated in another location (e.g. on a server)
- “`bzr bind <URL>`” and “`bzr unbind`” switch between the two



A Bit About Launchpad



- Launchpad is a free to use Web-based tool that supports open source software development
- Has several components: bug tracking, translations, code hosting, ...
- Designed to scale from the smallest (the shell script you write while listening to me) to the largest (Ubuntu) projects



Getting Started with Launchpad



- You can get source code without registering, but most interesting functions require registering an account

<https://launchpad.net/+login>

- Register SSH keys too, to allow uploading of code

<https://launchpad.net/people/+me/+editsshkeys>



Using Bazaar with Launchpad



- Register a project

`https://launchpad.net/projects/+new`

- Create a branch:

```
bzr init sftp://username@bazaar.launchpad.net/  
~username/yourproject/trunk
```

- and check it out:

```
bzr co sftp://username@bazaar.launchpad.net/  
~username/yourproject/trunk yourproject-trunk
```

(or push an existing branch)



Collaborating on Launchpad



- Other people can make and register their branches of your code, but only you can commit to your branch
- Launchpad has a concept of a “team”, a group of people, for this and other situations
- Teams can be created at

<https://launchpad.net/people/+newteam>



Collaborating on Launchpad



Once you've created a team, you can push a branch that all members of the team can commit to at

```
sftp://bazaar.launchpad.net/~yourteam/yourproject/  
trunk
```

(or reassign an existing branch to the team)



Why Use Launchpad?



- You can host Bazaar branches on more or less any web server – why should you use Launchpad?
- Get things for free:
 - Someone else looking after the server
 - Web views of your branch
 - Branch notifications (commit mails)
 - Simple access control through teams



Why Use Launchpad?



- A problem with DVCS is that of finding branches – Launchpad is a good place to look to find Bazaar branches
- If you want to host your code on your own server, you can still get Launchpad to mirror your code



Working with other projects



- It's very easy to get the mainline code for any project on Launchpad:

```
bzr get lp:mailman mailman-trunk
```

- But what if the project you're interested in doesn't use Bazaar?



Using Bazaar if upstream doesn't



- A unique feature of Launchpad is the code imports service
- Given CVS or Subversion repository details, converts the history to a Bazaar branch
- And *tracks* changes in the foreign repository



Using Bazaar if upstream doesn't



- This reduces the barrier to contribution
- It's *much* easier to maintain a patch against upstream in a Bazaar branch than as changes in your working tree
- There's also the bzd-svn plugin, which allows you to commit back to svn (but has different trade-offs)



Bazaar and other parts of Launchpad



- In this talk I've only really talked about the code hosting side of Launchpad
- There is integration with other parts of Launchpad, for example, you can associate a branch to the bug it fixes or specification of a feature it implements



Bazaar and Launchpad



- You don't have to use Launchpad to use Bazaar
- You don't have to use Bazaar to use Launchpad
- But the two parts add up to a greater whole



Coming Soon...

- Bazaar performance is a major focus right now
- Better and better foreign VCS support
 - bzd-svn already pretty good
 - bzd-hg, bzd-git?
- Built-in support in Launchpad for code reviews and PQM-style robots



Thanks!

Thanks to David Allouche, Blake Winton and assorted people from #bzd on freenode for comments on early versions of the slides

Any Questions?