Veracity

The Next Step in DVCS

Eric Sink

www.ericsink.com

No vendor crapola

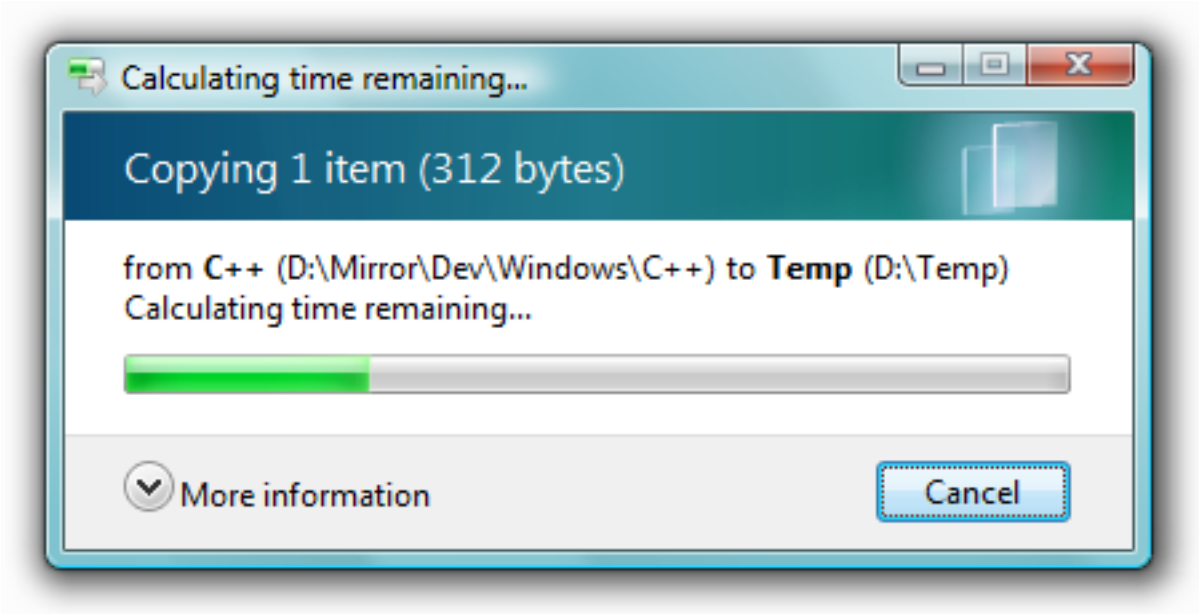
- Veracity is open source.
- I assume you already know what a DVCS is.
- There will be curly braces and semicolons in the slides.

5 Benefits of DVCS

- Fast
- Cruise Ship
- Moscow
- Dell
- Lego

Fast

- Developer operations, everything local

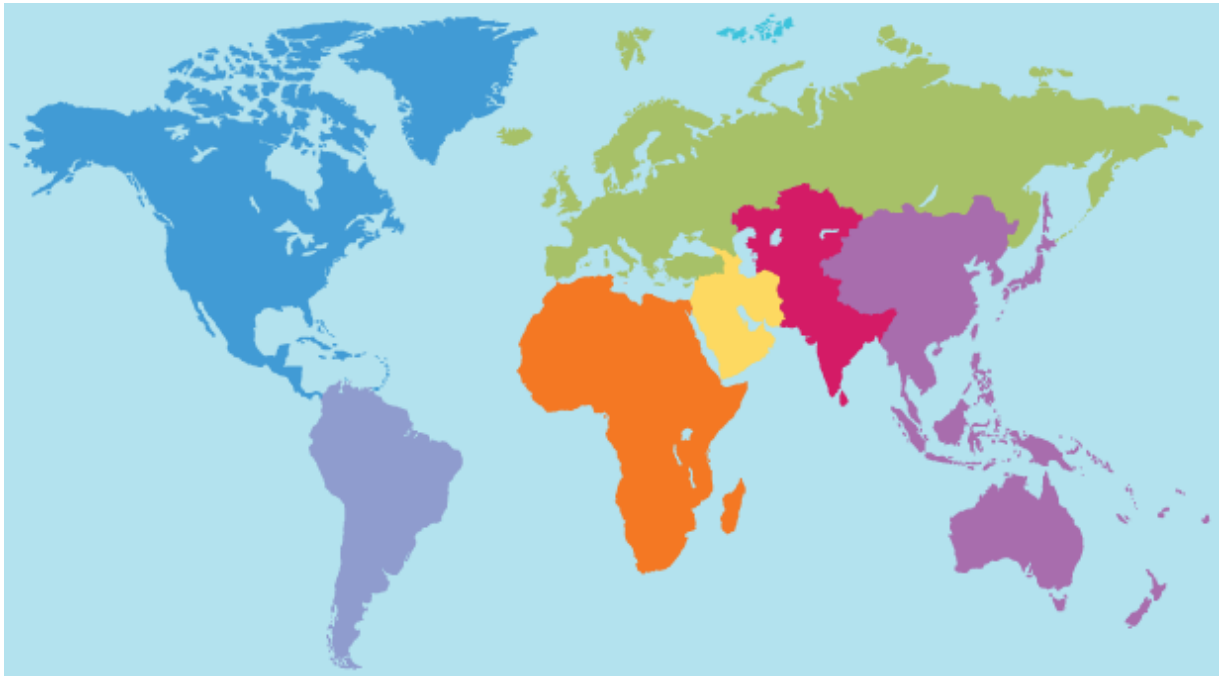


Cruise Ship

- Disconnected operation

Moscow

- Distributed teams, offshoring

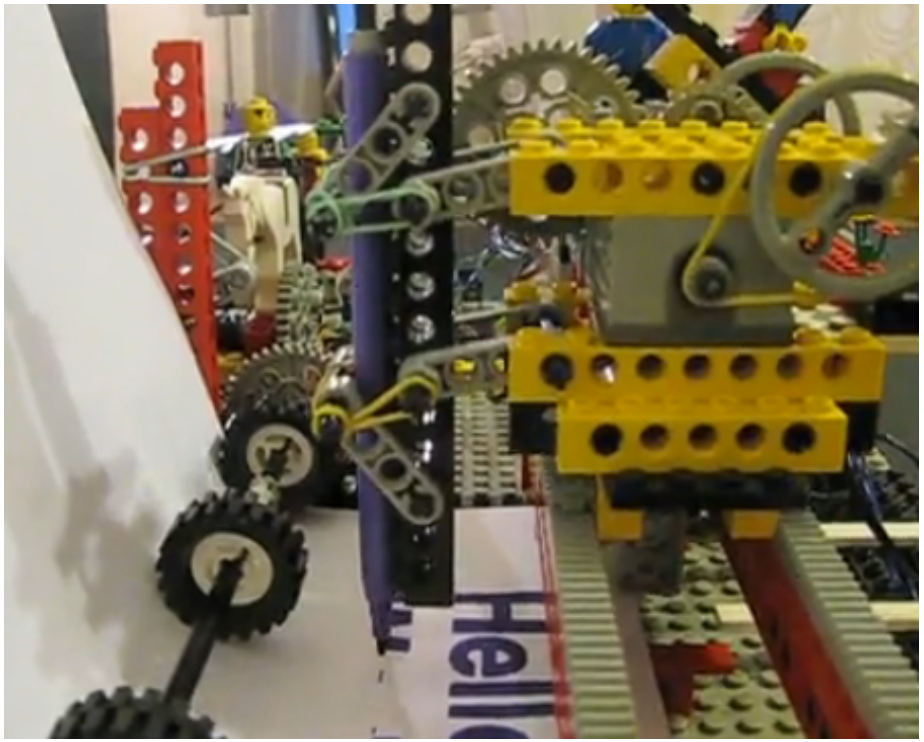


Dell

- Scale out instead of up

Lego

- Complex processes



So …

- Distributed source control tools are great

Luckily…

- Source control is the only thing software teams ever use.
- ☺

Other stuff

- Bug tracking
- Wiki
- Twitter-ish things
- Discussion forum
- Agile planning
- Support tickets
- Build tracking
- Requirements
- Test management
- Etc.

Integration

- The trend is toward seamless integration of all the tools that software teams use.

- Trying to mix centralized and decentralized gets just what you would expect.

Oil and Water ☹

Fast	Source code repo is fast, but I have to hit the server for every other operation.
Cruise Ship	I can checkin my code now. Later I'll update bug-tracking, wiki, etc. If I remember.
Moscow	Local server for DVCS, but for everything else, we still need dependable net access to a server in the home office.
Dell	And that server was more expensive than my house.
Lego	And our whole development process is dictated by quirky constraints of the central server.

What we need

- The benefits of DVCS
  - But for the non-source-code stuff
- Not just tree/filesystem data
  - But records and fields
- Changesets
  - Push, pull, merge

Other approaches

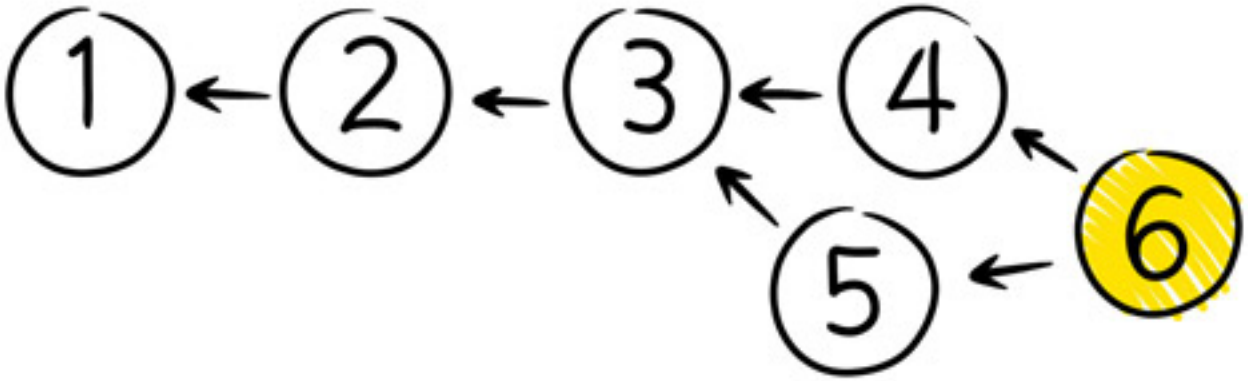
- Ditz, ticgit, git-issues, Bugs Everywhere
  - Store everything in the tree
  - Merge: Line-based source code merge
- Fossil (www.fossil.org)
  - Decentralized database
  - Merge: Latest timestamp wins

Veracity

- www.sourcegear.com/veracity
- Apache License Version 2.0

- Decentralized Database
  - Template-driven merge
  - “zing”

Merging



- Automerge using rules specified in the template


```
{
    "version" : 1,
    "rectypes" :
    {
        "item" :
        {
            "fields" :
            {
                "val" :
                {
                    "datatype" : "int",
                }
            }
        }
    }
}
```

Correctness

- Attitudes about automerge from source control
- A merge is correct if the template defines it to be
- It is possible to define a template such that the merge cannot fail
- Log of changes made

Two basic problems to solve

- Conflicting changes
- Constraint violations

Non-Conflicting Changes

- Two people add unrelated records
- Two people modify a record, different fields
  - If template allows this
  - Example


```
// simple template
var t =
{
    "version" : 1,
    "rectypes" :
    {
        "item" :
        {
            "fields" :
            {
                "title" : { "datatype" : "string" },
                "priority" : { "datatype" : "string" }
            }
        }
    }
};
```

sourcegear

```
// setup a new repo
reponame = sg.gid();
sg.new_repo(reponame, false);
repo = sg.open_repo(reponame);
zs = new zingdb(repo, sg.dagnum.TESTING_DB);
ztx = zs.begin_tx();
ztx.set_template(t);
ztx.commit();

// initial version of the record
ztx = zs.begin_tx();
zrec = ztx.new_record("item");
recid = zrec.recid;
zrec.title = "car stalls";
zrec.priority = "low";
original_state = ztx.commit().csid;
```

sourcegear

```
// two people modify the record
ztx = zs.begin_tx(original_state);
zrec = ztx.open_record(recid);
zrec.title = "car stalls, and then EXPLODES";
ztx.commit();

ztx = zs.begin_tx(original_state);
zrec = ztx.open_record(recid);
zrec.priority = "URGENT";
ztx.commit();

// merge and get both changes
zs.merge();
print(sg.to_json(zs.get_record(recid)));

repo.close();
```


```
"priority" : "URGENT",
"recid" : "g38008c2ef32d457ea492f80066c47963b1a06dc4936611dfba6
",
"rectype" : "item",
"title" : "car stalls, and then EXPLODES",
"history" :
[
    {
```

Conflict

- Two people modify same record, same field
- Template can specify a resolution


```
"fields" :
{
    "latest_news" :
    {
        "datatype" : "string",
        "merge" :
        {
            "auto" : [ { "op" : "least_recent" } ]
        }
    }
}
```


```
"fields" :
{
    "estimate" :
    {
        "datatype" : "int",
        "merge" :
        {
            "auto" : [ { "op" : "max" } ]
        }
    }
}
```


```
"fields" :
{
    "severity" :
    {
        "datatype" : "string",
        "constraints" :
        {
            "allowed" : ["annoying", "data loss", "smoke"]
        },
        "merge" :
        {
            "auto" : [ { "op" : "group:qa" } ]
        }
    }
}
```

Unique constraints

- Uniqueness in a distributed system is a classic problem
- GUIDs

- g6c0ae12907dd4789b880a68026ac498f5df1aa2c935611dfbbd860fb42f09aca

- Friendly bug IDs


```
"fields" :
{
    "val" :
    {
        "datatype" : "string",
        "constraints" :
        {
            "unique" : true
        }
    }
}
```

Uniqify

- Automatically resolve unique constraint problems on merge by modifying one of the records.

- Which one to modify?
- How to modify it?


```
"val" :
{
    "datatype" : "int",
    "constraints" :
    {
        "unique" : true
    },
    "merge" :
    {
        "uniqify" :
        {
            "which" : "last_modified",
            "op" : "add",
            "addend" : 5
        }
    }
}
```

An ounce of Prevention

- Generate friendly ids that are *likely* to be unique
- Then uniqify during merge when necessary

Example

- userprefix + digits
- E0001


```
"id" :
{
    "datatype" : "string",
    "constraints" :
    {
        "unique" : true,
        "defaultfunc" : "gen_userprefix_unique"
    },
    "merge" :
    {
        "uniqify" :
        {
            "op" : "inc_digits_end",
            "which" : "last_created"
        }
    }
}
```

Followups

- Lots more cool stuff in Veracity
  - sourcegear.com/veracity
  - www.ericsink.com

- @eric_sink
- eric@sourcegear.com
- SourceGear booth