Version Control Systems

Aims

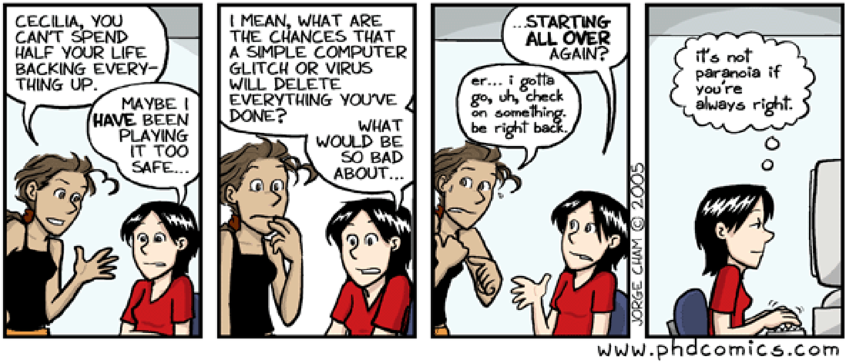
- Help identify problem that can be solved.
- Introduce basic concepts of version control.
- Explain why various technologies exist, and which you should choose.

When you need version control

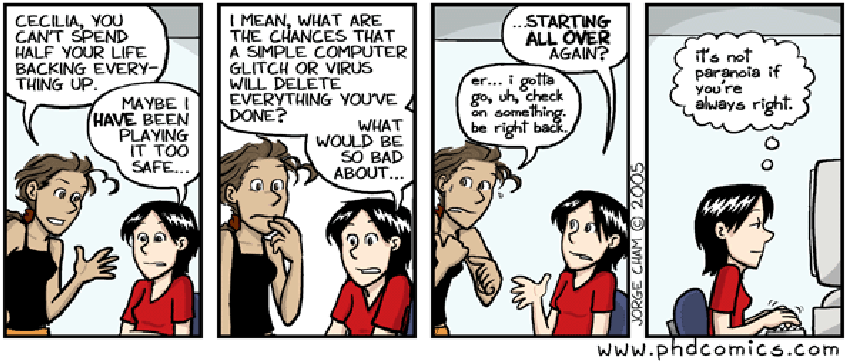
- Complex documents, built up over time.
- Multiple collaborators (or even just multiple machines).
- Multiple versions which ‘co-evolve.’
- Reproducibility (‘snapshots’).

Four Evolutionary Stages

Stage 0: Not backing up


CECILIA, YOU CAN'T SPEND HALF YOUR LIFE BACKING EVERY-THING UP.
MAYBE I HAVE BEEN PLAYING IT TOO SAFE...
I MEAN, WHAT ARE THE CHANCES THAT A SIMPLE COMPUTER GLITCH OR VIRUS WILL DELETE EVERYTHING YOU'VE DONE?
WHAT WOULD BE SO BAD ABOUT...
...STARTING ALL OVER AGAIN?
er... i gotta go, uh, check on something. be right back.
it's not paranoia if you're always right.
JORGE CHAM © 2005
www.phdcomics.com

Stage 0: Not backing up


CECILIA, YOU CAN'T SPEND HALF YOUR LIFE BACKING EVERYTHING UP.
MAYBE I HAVE BEEN PLAYING IT TOO SAFE...
I MEAN, WHAT ARE THE CHANCES THAT A SIMPLE COMPUTER GLITCH OR VIRUS WILL DELETE EVERYTHING YOU'VE DONE?
WHAT WOULD BE SO BAD ABOUT...
...STARTING ALL OVER AGAIN?
er... i gotta go, uh, check on something. be right back.
it's not paranoia if you're always right.
JORGE CHAM © 2005
www.phdcomics.com

DON'T DO THIS

Stage 1: Manual copies

"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDIDICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012

Stage 1: Manual copies

Flaws:

- Manual = fallible.
- Backup: Copies of copies.
- Labelling.

Stage 1: Manual copies

Flaws:

- Manual = fallible.
- Backup: Copies of copies.
- Labelling.

We need **metadata** - datestamps, annotations, attribution.
And **tools** - make this stuff quick and easy!

Aside: ‘Cloudy’ technologies

Trade off — convenience vs control.
Good for:

- Small docs, frequently updated across multiple locations (e.g. to-do list).
- Basic backups of items unlikely to evolve (photos, etc).

Aside: ‘Cloudy’ technologies

Problems:

€ Versioning is all automated - can’t choose sensible ‘checkpoints’ to mark out.

€ Collaboration is still broken, unless you’re working on very simple docs.

Aside: ‘Cloudy’ technologies

Dropbox

Problems:

- Versioning is all automated - can’t choose sensible ‘checkpoints’ to mark out.
- Collaboration is still broken, unless you’re working on very simple docs.

NEED MORE METADATA

Stage Two

Centralised version control

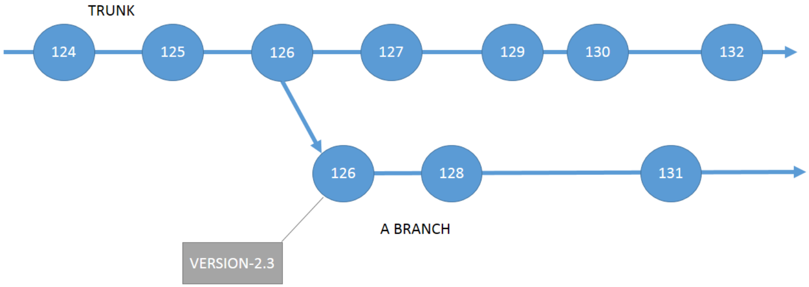
e.g.

- ‘Concurrent Versions System’ (CVS, now defunct).
- ‘Subversion’ (SVN).

SUBVERSION

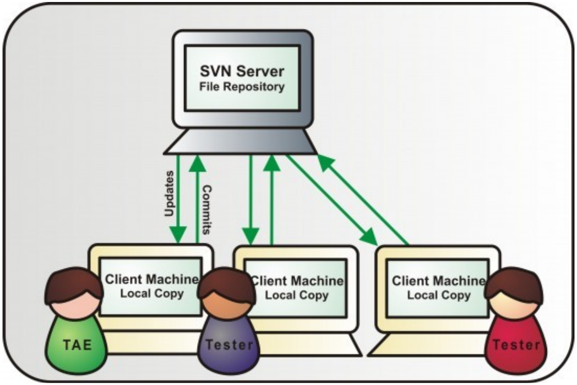
Basic concepts, 1

Record an annotated history of change sets.



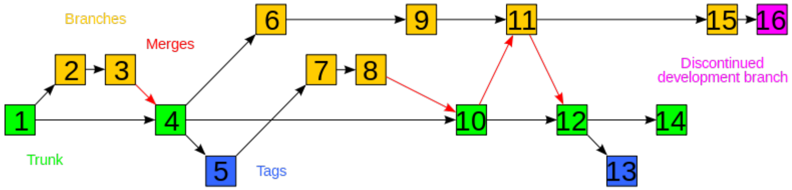
- Trunk, branch
- Parents, ancestors

Basic concepts, 2



- Repository
- Checkout
- Commit /

Basic concepts, 3

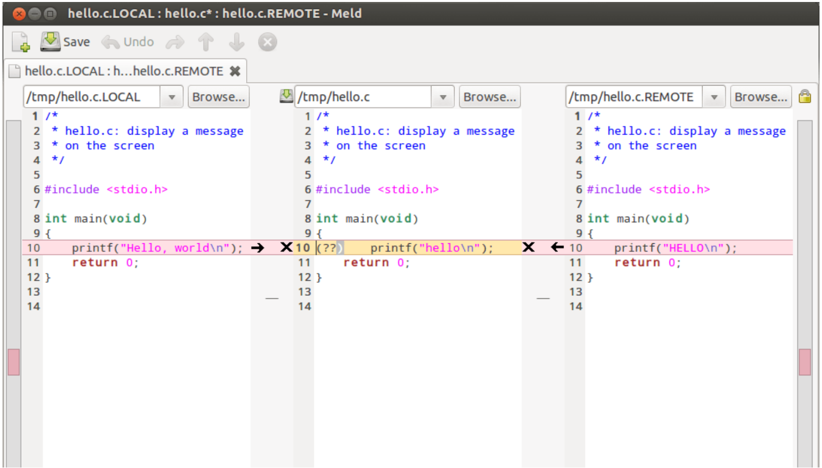


Merging

In simple cases, merges are automatic! Tree-records allows us to build the new combined version.

Basic concepts, 3

Manual merging: When conflicts exist, we have the info and tools to manually resolve them.



```
/*
 * hello.c: display a message
 * on the screen
 */

#include <stdio.h>

int main(void)
{
    printf("Hello, world\n");
    return 0;
}
```

```
/*
 * hello.c: display a message
 * on the screen
 */

#include <stdio.h>

int main(void)
{
(??)    printf("hello\n");
    return 0;
}
```

```
/*
 * hello.c: display a message
 * on the screen
 */

#include <stdio.h>

int main(void)
{
    printf("HELLO\n");
    return 0;
}
```

Distributed VCS

1986 – early 2000’s: Why would you make this any more complex? This works.

Distributed VCS

1986 – early 2000's: Why would you make this any more complex? This works.

INTERWEBS

Distributed VCS

1986 – early 2000's: Why would you make this any more complex? This works.

INTERWEBS

(See e.g. visualised history of Python, https://www.youtube.com/watch?v=_cNBtDstOTmA)

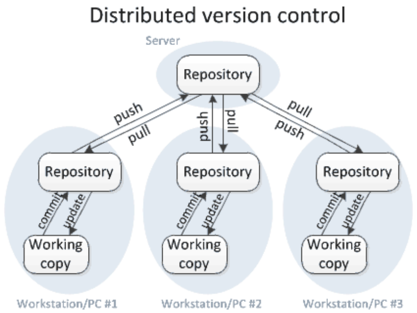
Centralised doesn’t scale

- Many collaborators.
- Cannot check-in half-finished work to master.
- Cannot keep track of a branch for every collaborator.

Centralised doesn’t scale

- Many collaborators.
- Cannot check-in half-finished work to master.
- Cannot keep track of a branch for every collaborator.
- Resort back to hybrid of central copy under version control, with many local, manual backups for intermediate work.

The distributed model



- Everyone has their own mirror, or *clone* of the repository.
- Changes are distributed via *pushes* and *pulls*.

Distribute!

Benefits for you:

- More flexible. Allows different workflows and collaborative behaviour etc.
- Can *commit* offline, sync later.
- Talk to me later if you want the details.

So which should I use?

So which should I use?

At this stage, git and mercurial are functionally equivalent — but git has won the majority mindshare, therefore: better support, better chance of collaborators using same system, etc.

Summary

- Version control helps with:
  - Backups
  - Reproducibility
  - Comparing arbitrary historical versions.
  - Maintaining multiple live versions.
- Lots of free services and material online to help you out.
- Bit of a learning curve at first - but payoff is large in long-run. (And now you have a headstart!)

Advanced Reading

To start, google 'git intro', etc. Then. . .

- Git for Computer Scientists
  http://eagain.net/articles/git-for-computer-scientists/

- Understanding Git Conceptually
  http://www.sbf5.com/~cduan/technical/git/

- Understanding the Git Workflow
  https://sandofsky.com/blog/git-workflow.html