Development workflow

Computing Methods for Experimental Physics and Data Analysis

L. Baldini

luca.baldini@pi.infn.it

Università and INFN–Pisa

Compiled on September 28, 2019

Introduction

- Most of us would agree that research should be correct *and* reproducible
  - "Reproducible": *able to be shown, done, or made again*[1]
- Reproducible research is data analysis that starts with the raw data and arrives at the same answers
- Software plays a paramount role in modern experimental physics
  - Therefore a crucial ingredient in reproducibility of research
  - It should be developed under controlled conditions, just like the experiments are performed
- Often times physicists are better at correctness than reproducibility
- Scientists are famous for producing poor-quality software
  - The community is lagging behind on the best practices that are ubiquitous among professional software development
  - (And there are many good reasons why that is the case)
- Spoiler: today it's gonna be boring, but it will get better!

[1] Source: the Cambridge Dictionary

A few initial questions

- Consider the last piece of analysis software you have written
  - Would somebody else able to use it without your help?
  - Would it run as is on somebody else's computer?
  - Would you be able to adapt it for somebody else's computer?
  - Would you be able to run it on your computer a year from now?
- Not even mentioning efficiency
  - Could your software run significantly faster?
  - And if so: would you care?
- And now take the last paper that you have read
  - Would it be possible to re-do the analysis (obtaining the same results)?
  - Have the raw data been preserved in a readable format?
  - Is the analysis software still available?
  - Is there still somebody around understanding what the software does?
  - Are there machines around that can run the analysis software?
  - Is the software even version-controlled?
  - (You would be surprised by the answers)
- Standards sometimes help. . .
  - e.g., the FITS standard for astronomical data
- . . . but the often hinder performance
  - e.g., compared to ROOT, FITS I/O sucks by any reasonable metrics

Developing code collaboratively

- Most modern experiments are run by large teams
  - The big LHC Collaborations count thousands of members
- How do you handle development in such an environment?
  - How do you distribute the source code?
  - How do you know which version is any single person running?
  - How do hundreds of people develop the same code concurrently?
  - How do you make sure that your last awesome change isn't breaking everybody else's code?
- And even if you work in a small team
  - How do you exchange code with your collaborators?
  - How do you collect feedback?
  - How do you distribute updates?
- And even if you are working by yourself
  - Why is the thing giving a different answer with respect to the last time I run it, last week?
  - And which is the correct answer?

The first key concept: version control

- How do you handle version control (in order of severity)?
  - What the hell is version control?
    - Welcome to this class!
  - I heard about such thing, but I don't need it
    - If you really think so, you do need it but don't know yet!
  - Every once in a while I create a copy of a file with a `_vn` appended
  - Or I periodically create an archive and attach a version number to it
    - And how is that helpful when something goes wrong?
  - I created my own custom-made system
    - mhhh. . . suspicious—git was written by Linus Torvalds
  - I do use a professional tool for the job (e.g., git)
    - Congratulations: you are a person of good taste!
- "Version control": *A component of software configuration management, version control, also known as revision control or source control, is the management of changes to documents, computer programs, large web sites, and other collections of information*[2].
  - Collect meta-data whenever the code is changed
  - What, when, by whom and why?
  - Freeze forever a specific configuration of the code-base

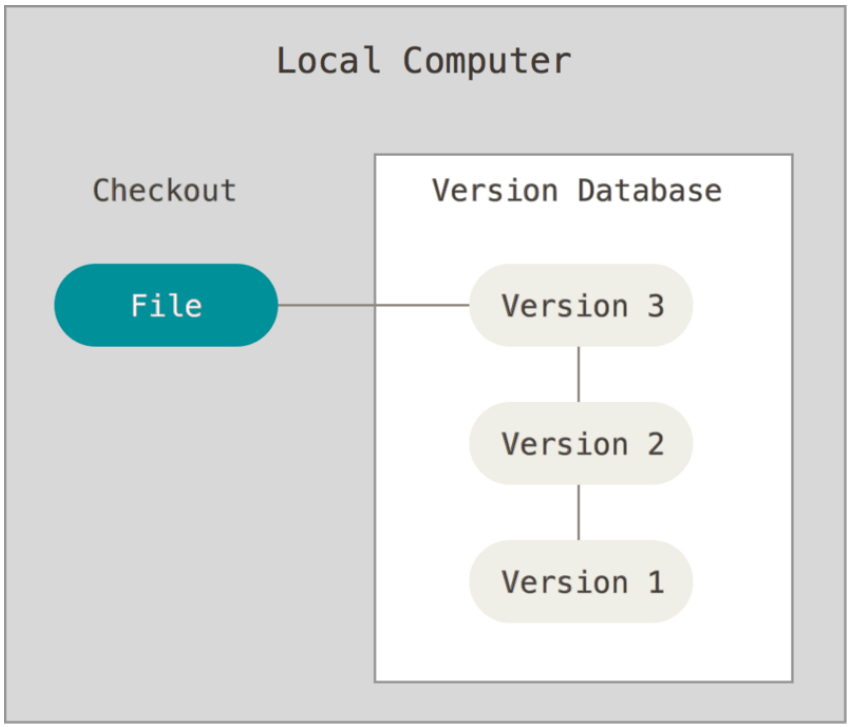
[2] Wikipedia

Basic terminology

- Repository: the place where files' current and historical data are stored
- Revision or version: the state at a point in time of the entire tree in the repository
- Clone: creating a repository containing the revisions from another repository
- Working copy: a local copy of files from a repository at a specific revision
- Checkout: create a local working copy from the repository
- Change or diff: a specific modification to a set of files under version control
- Commit: write the changes made in the working copy back to the repository

Local version control systems

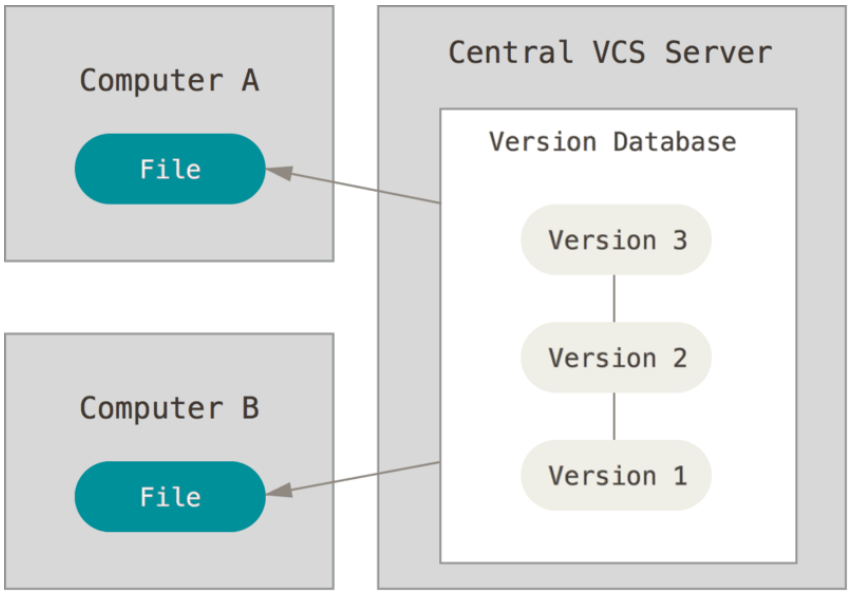
e.g., RCS



- Keeps differences between revisions in a local database
  - Neat idea, as it saves disk space!
- Can recreate what any file looked like at any point in time

Centralized Version Control systems

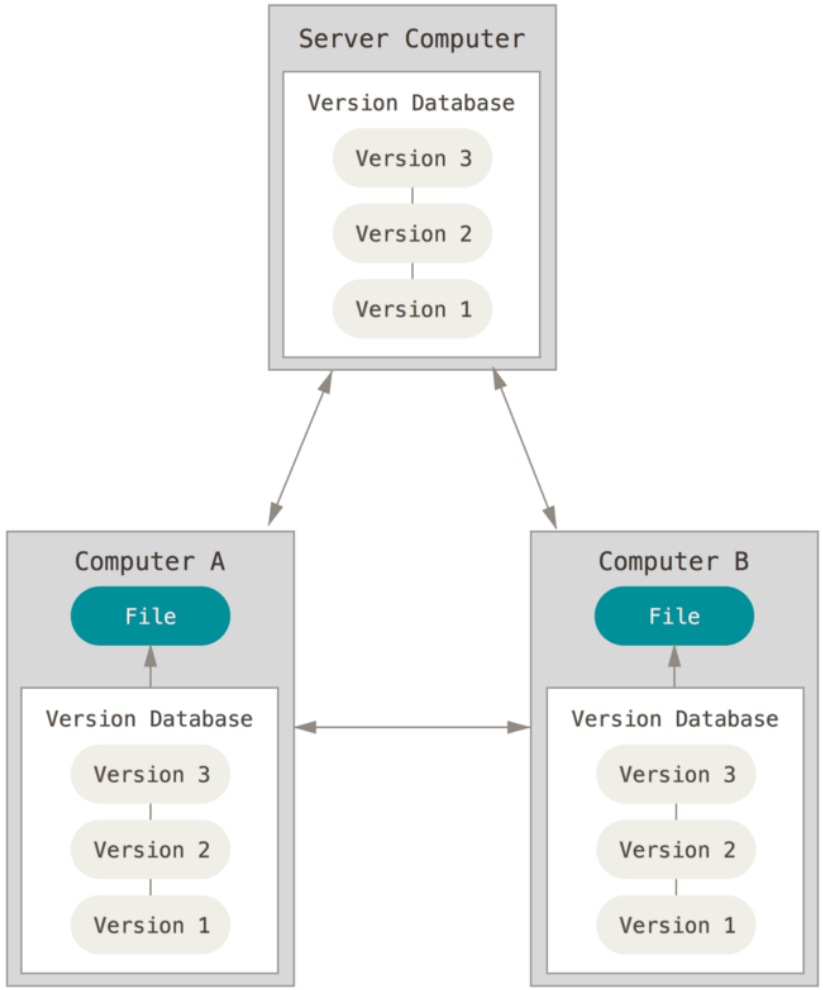
e.g., CVS, Subversion



- Single server containing all the versioned files
  - Clients can check out the files from the repository
- Basic work-flow
  - Check out a local working copy from the remote server
  - Modify the working copy
  - Commit the changes back to the repository
- Most popular model through most of the '90

Distributed version control system

e.g., git, mercurial

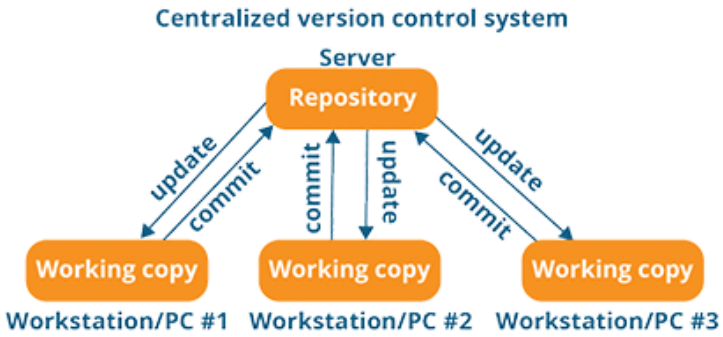


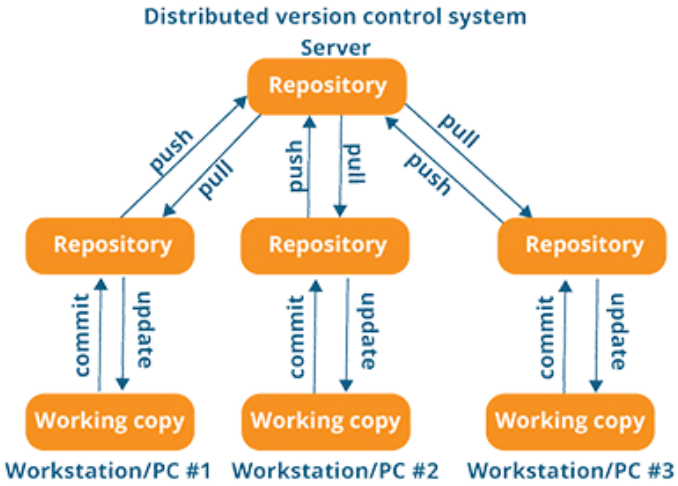
- Clients fully mirror the repository, including its full history
- Allows for a much richer variety of work-flows

Versioning single files vs. the entire repository

- Old VCS only tracked modification on a file-by-file basis
  - i.e., CVS assigns revision numbers to the single files
- All modern VCS track a whole commit as a new revision
  - i.e., revisions are assigned to the repository
- It makes a lot of sense to version the entire repository
  - Versioning single files can give a cozy feeling...
  - ...but when files interact with each other you need to know the status of the entire repository to reliably predict the output
- Don't try and do the work that the VCS is supposed to do
  - Do not keep around multiple copies of the same file with a different suffix—the VCS is keeping track of the changes
  - Do not copy and paste a block of code, comment the original and edit the second—this will make diffs harder to understand

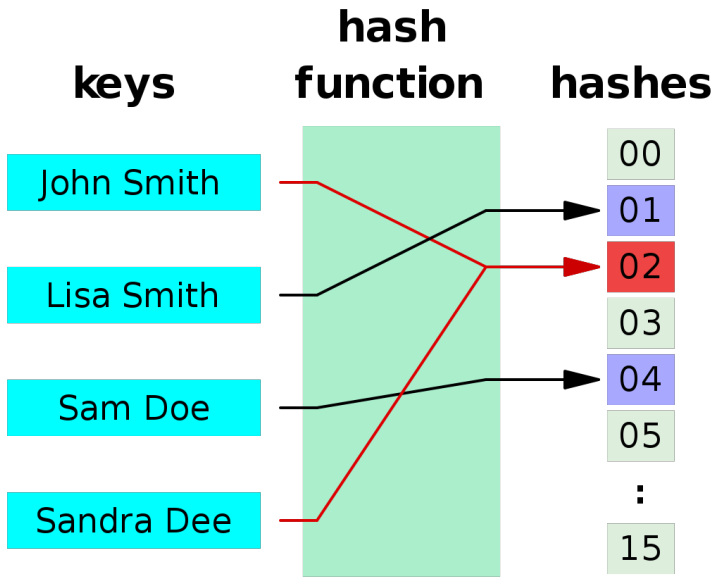
Centralized vs. distributed VCS





- The centralized VCS work-flow is *linear*
  - Subversion assigns a progressive number to the repo at each commit
- In a distributed system the work-flow is inherently *non-linear*
  - Any one given local repository is not ahead or behind any other repository—just different
- But then how do we assign revision in a distributed VCS?

Digression: hash functions



- Ever heard of checksums? Or short versions of doodle links?
- A hash function maps data of arbitrary size to fixed-size values
  - e.g., anything to an integer
- Good properties for a hash function
  - Deterministic
  - Uniform in the image space (minimize collisions)
  - Fast to calculate (and, possibly, hard to invert)

Hashing in Python

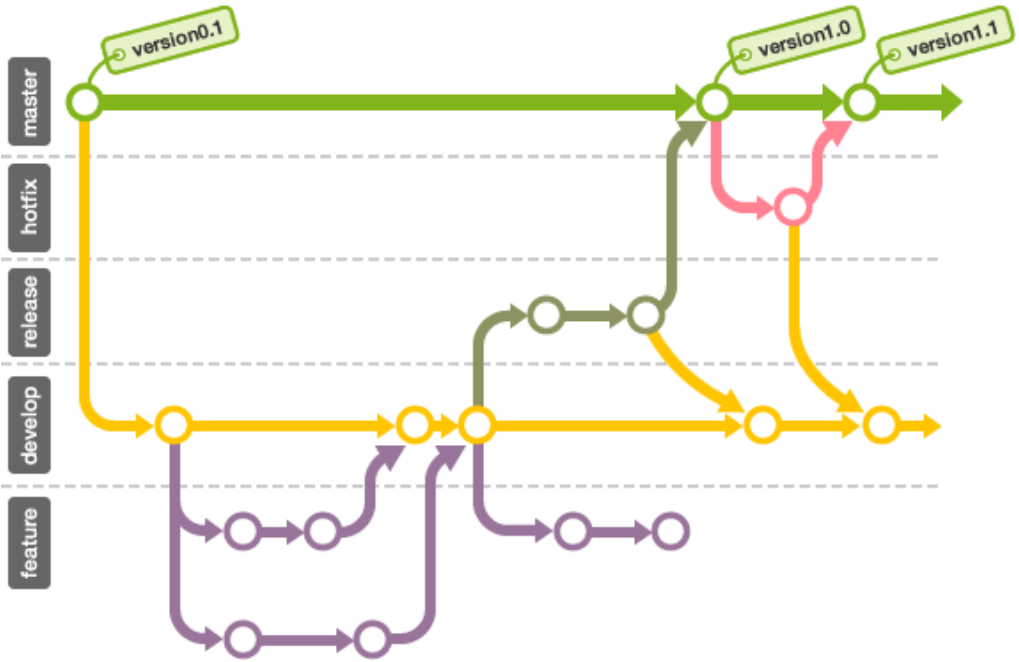
https://bitbucket.org/lbaldini/programming/src/tip/snippets/hashing.py

```
1  print(hash(3))
2  print(hash(3.))
3  print(hash(3.001))
4  print(hash('hello'))
5  print(hash('Hello'))
6
7  [Output]
8  3
9  3
10 2305843009213443
11 6550031156960066471
12 7799499919494320088
```

- Every *immutable* object is hashable in Python
  - Each type gets its own algorithm
- And why I am even bringing this up?
  - Because in distributed VCS each commit gets its own has

```
1  [lbaldini@nbbaldini slides]$ git rev-parse HEAD
2  e0a170d789865179d99506c9f067f3875292524b
```

More terminology



- Branches: alternative paths where more copies of the same files develop in different ways independently
- Master, or trunk, or tip: the unique line of development that is not a branch
- Merge: application of two sets of changes to a set of files
- Conflict: changes to the same file by two or more developers that the system is unable to reconcile

Basic git commands

Try `git --help`

```
start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout   Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects
```

A simple but realistic work-flow

1. Clone your repository
2. Create a new branch and check it out
3. Do all you have to do
4. Add the new files, commit your modified files
5. Create a pull request
6. Have your pull request reviewed
7. Merge the branch on the master
8. Tag the master

A few suggestions

- Pick a sensible name for your branches
  - Try and express your *intent*
  - Examples of bad names: `my_branch`, `new_branch`, `awesome_branch`
  - Examples of reasonable names: `update_data_format`, `fix_issue_2376`
- Try and isolate related changes in the smallest possible subset
  - Attack one problem at a time
  - Do not mix unrelated issues in the same pull request
  - The smallest the pull request, the easiest the review
- Do not try and do what the VCS is suppose to take care of
  - Do not commit to the repository multiple versions of the same file—the VCS is perfectly capable of tracking the changes
  - Do not copy and paste chunck of code, comment the old version and tweak the new—this will make diffs harder to read
- VCS are good at handling text files
  - Do not commit large binary files (e.g., data sets) to your repository—it will stay there forever

Logistical remarks

- Which VCS should I use?
  - Everybody else is using git, these days—you might as well stick to that
- Hosting: shall I set up a server myself or use a hosting service?
  - This is a no brainer: use a service (e.g., github, gitlab, bitbucket)
- What shall I use version control for?
  - Your thesis
  - Your papers (or lab assignments)
  - Your webpage (if you have one)
  - The invitations for your wedding
  - Really: everything you care about!
- Shall I care about the license?
  - Definitely: make easy for others to use and adapt your code (unless you have good reasons not to)
  - GNU General Public License v3.0 is a good choice for code
  - Creative Commons Attribution-ShareAlike 4.0 is a good choice for documents

References

- https://en.wikipedia.org/wiki/Version_control
- https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control
- https://en.wikipedia.org/wiki/Hash_function
- https://git-scm.com/
- https://github.com/
- https://bitbucket.org/
- https://about.gitlab.com/