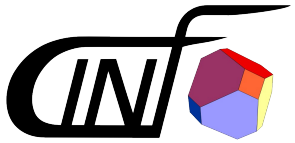
# xkcd on git


Mouse over:

If that doesn't fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of “It's really pretty simple, just think of branches as...” and eventually you'll learn the commands that will fix everything.

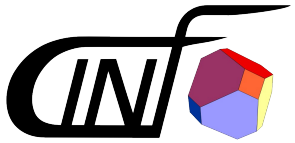
Source: http://xkcd.com/1597/



Center for Individual Nanoparticle Functionality

- Introduction
- Git data structures part 1
- Git data structures part 2
- Branches
- Network structure
- Summary

# svn to git
# 15+ years of new concepts

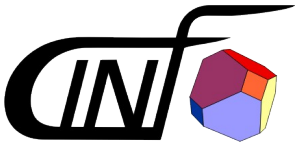


Center for Individual Nanoparticle Functionality

- **Introduction**
- Git data structures part 1
- Git data structures part 2
- Branches
- Network structure
- Summary

# svn to git
# 15+ years of new concepts

# A bit about me

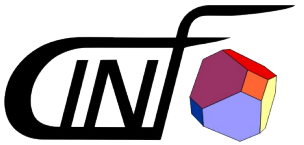


- Kenneth Nielsen, PhD from CINF
  - knielsen@fysik.dtu.dk
  - git clone https://github.com/KennethNielsen/presentations
- git (and github) user for 3 years
- Co-maintainer of SoCo (2 years)
  - pyg3t, poproofread
- Love git (almost as much as Python)
  - python-tips
- A bit of a gear head
  - Like to learn about tech

# Credits

- “Introduction to Git with Scott Chacon of GitHub”
  - https://www.youtube.com/watch?v=ZDR433b0HJY
  - Quite simply the best introduction I have seen
  - I use some figures from there (with permission)
- “Tech Talk: Linus Torvalds on git”
  - https://www.youtube.com/watch?v=4XpnKHJAok8
  - Original creator of git
  - Strong opinions (and language)
  - Distribution and trust
- “Pro Git” by Scott Chacon, for figures

# More background than tutorial



- Talk of difficult transition
  - Here to share my love for the tool
- Todays presentation is not a tutorial
  - Although I would be happy to do one
- Experienced learners

- Other challenges
  - Why does git insist on being that much different?
  - Learn git in the context of svn (bad experience)

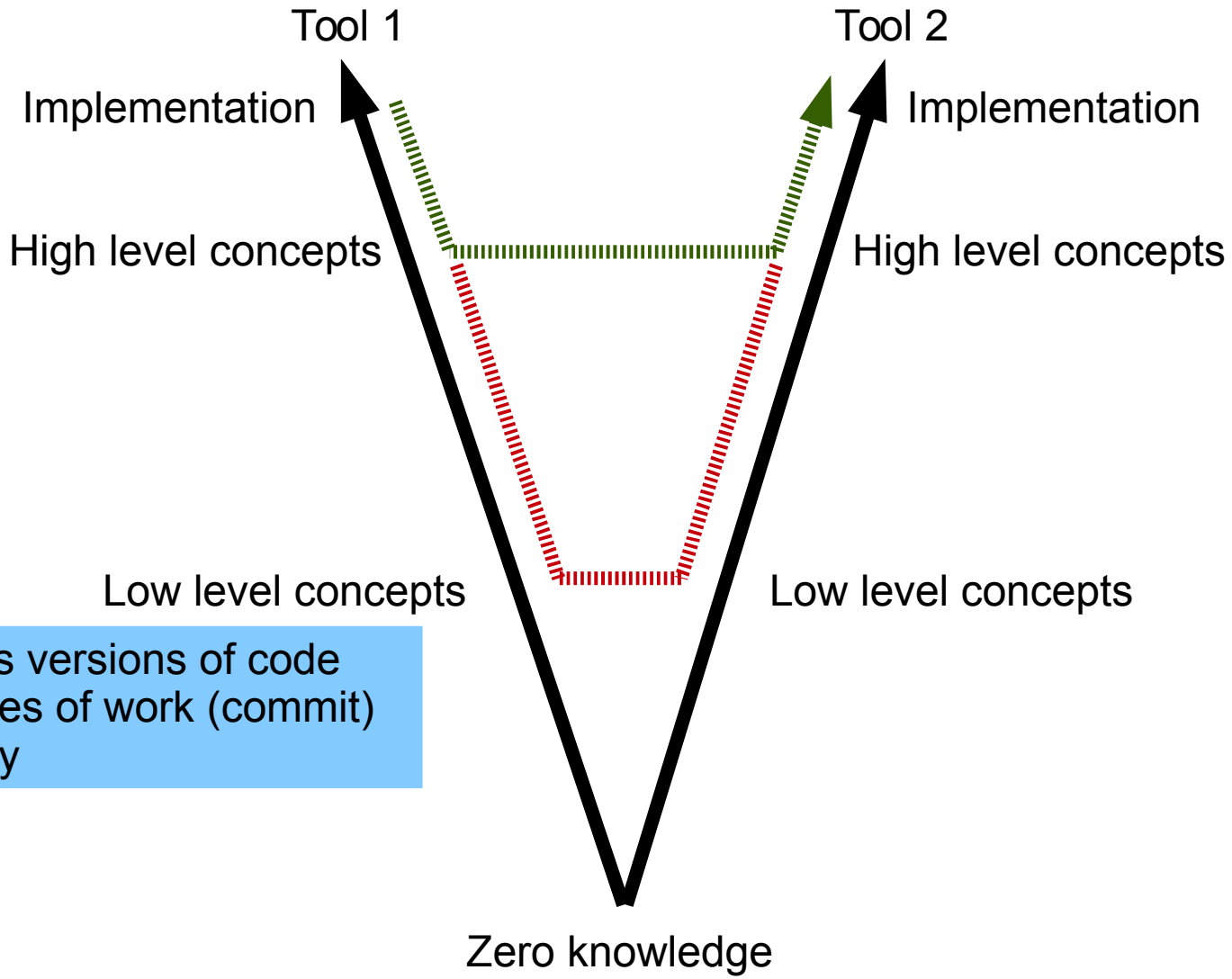
# Git != SVN

- Network structure
- How git thinks about its data
- Internal data structure
- Workflows

!=

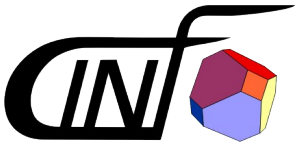
- Network structure
- How SVN thinks about its data
- Internal data structure
- Workflows

# Learning tool number 2

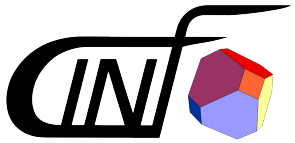



- Records versions of code
- Packages of work (commit)
- A history

# Common questions



- Why do I need **my own** repository on Gitlab?
- What do you mean, gits history is a tree?
- Use branches all the time? Does't that take up a lot of space and clutter up the list of branches for everyone else?
- What do you mean “add” and “checkout” does something different than in SVN
- “Fast forward merge”, speak english dammit!
- Why can't it just be more like SVN? I like SVN, I know SVN, git sucks!
- … <your question goes here> ?

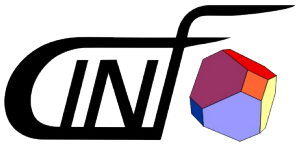


Center for Individual Nanoparticle Functionality

- Introduction
- **Git data structures part 1**
- Git data structures part 2
- Branches
- Network structure
- Summary

# svn to git
# 15+ years of new concepts

# How does VCS' think about their data



- Affects work flows
- Has implications for performance and usability
  - Operations on files vs. entire archive
  - Branching and merging
- Fundamentally different between SVN and git

- **SVN is a file based delta storage system**
  - Tracks files and stores metadata
- **Git is snapshot based**
  - Tracks snapshots of the entire repository

# File based delta storage (SVN)

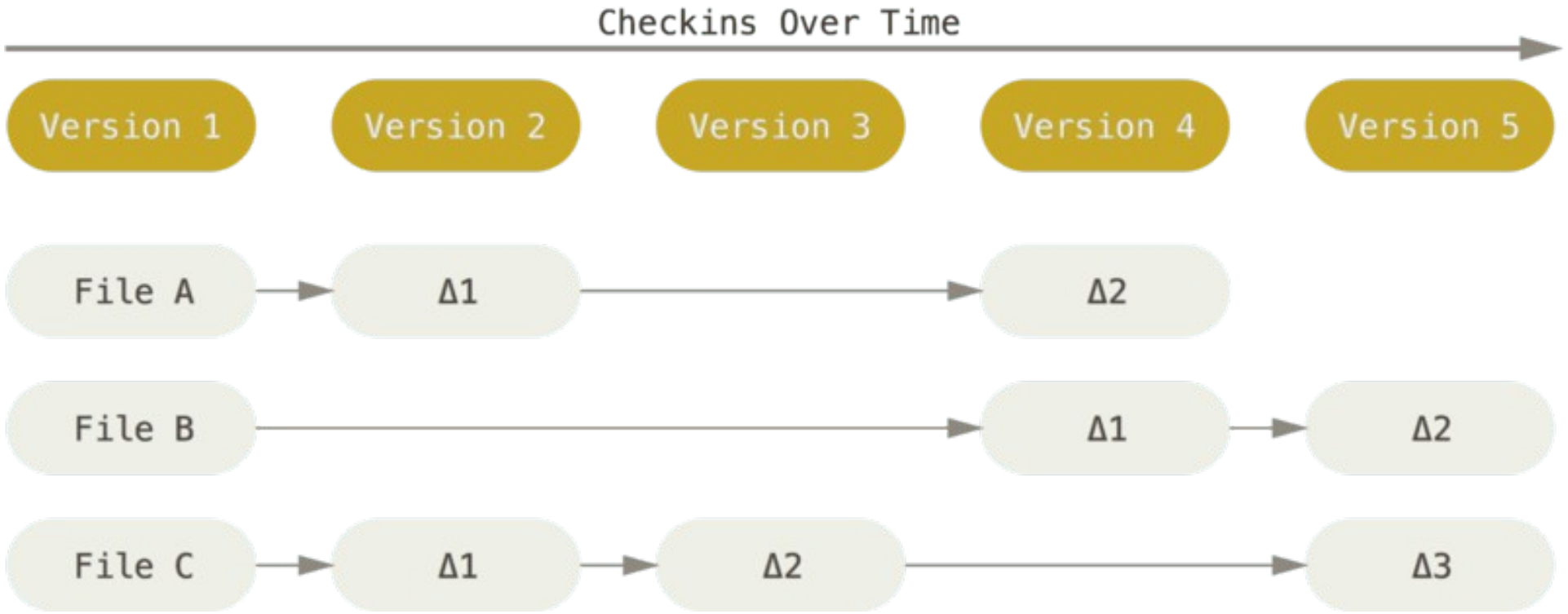



Figure from “Pro Git” by Scott Chacon, https://git-scm.com/book/en/v2

# Snapshot based (git)

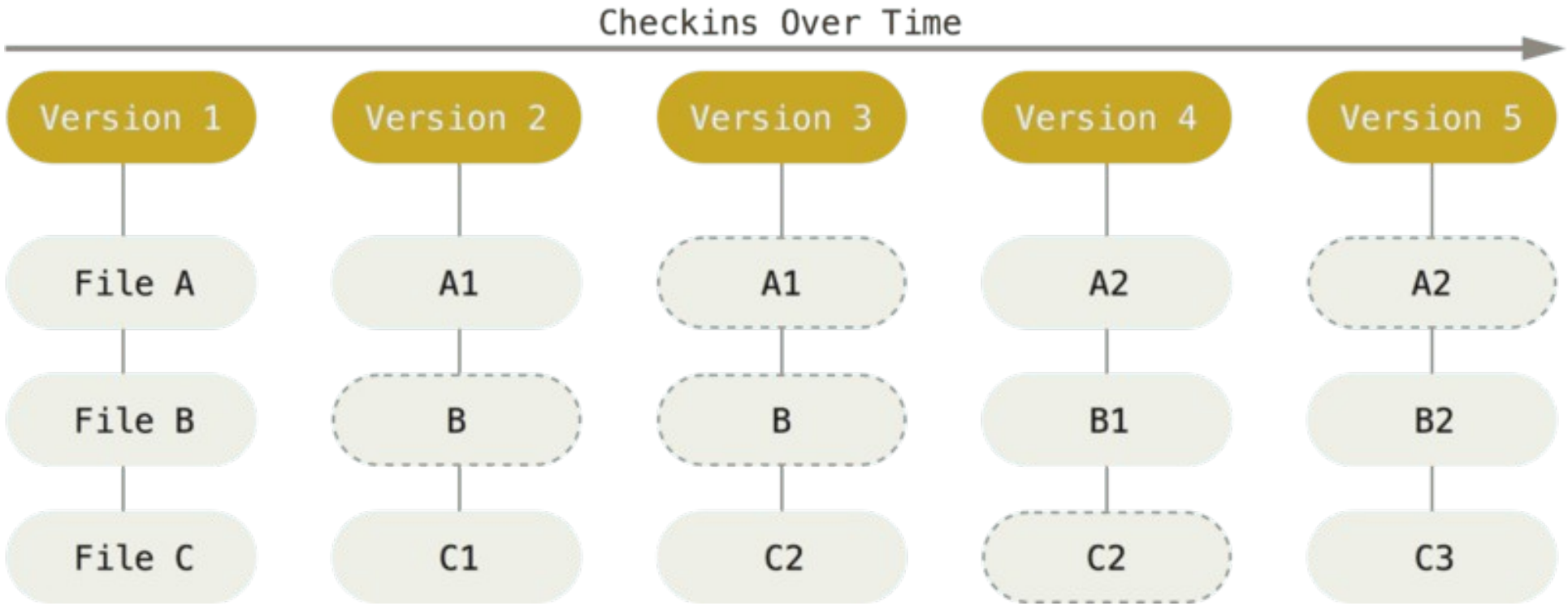
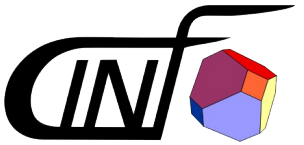


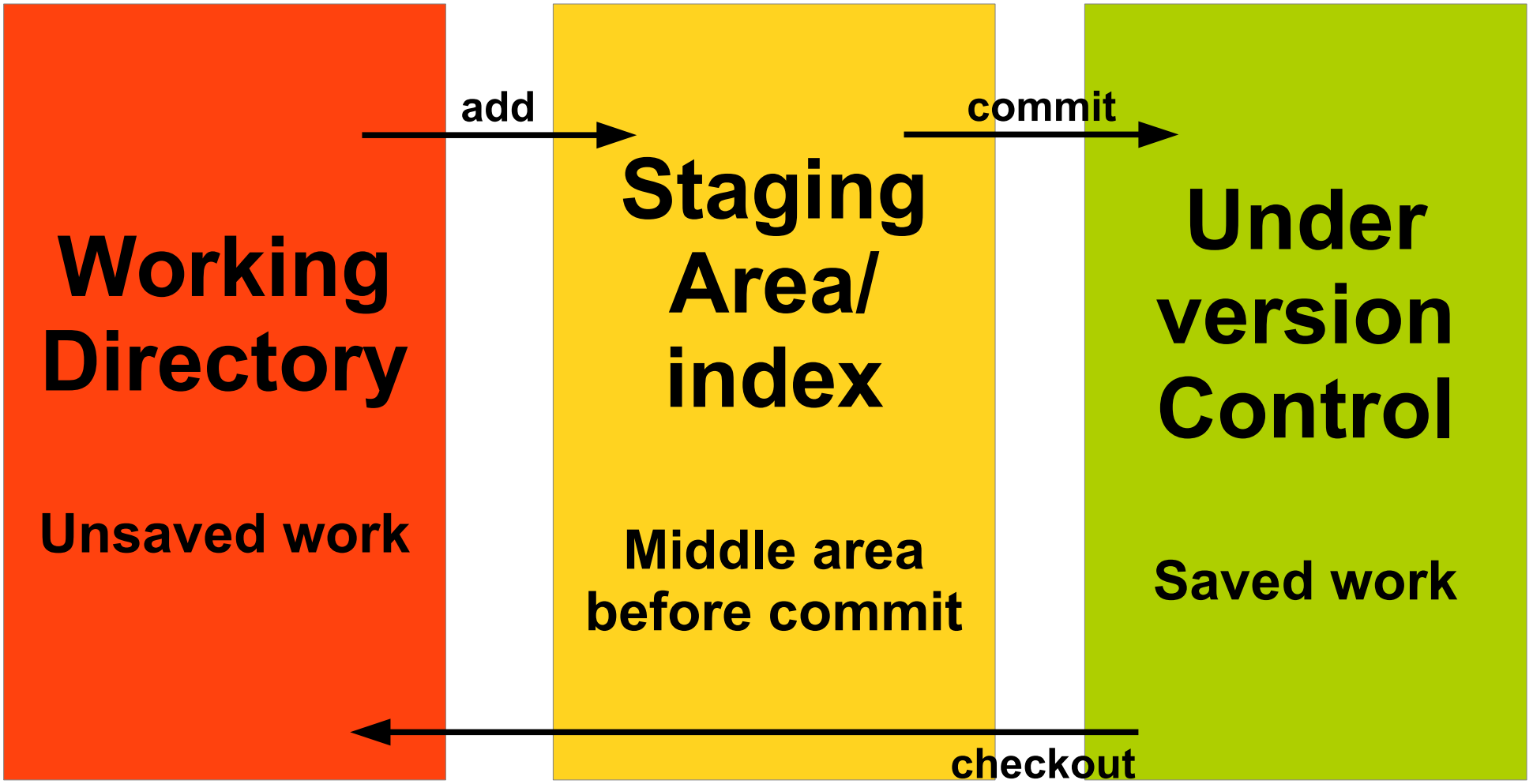

Figure from “Pro Git” by Scott Chacon, https://git-scm.com/book/en/v2

# “add” means something completely different

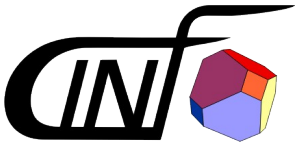


- SVN tracks files
  - “add” means “Here is a new file that I want you to track”
- Git does not
  - Uses the “add” verb for something completely different
  - “add” to staging area (middle area before commit)
  - “add” to next commit

# The staging area

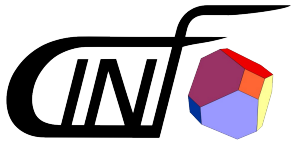


# Why the extra step?



- Allows for crafted commits
- Selective commiting
- Choose to only stage certain files
- Choose to only stage certain diff chunks

- Can be circumvented
  - git commit -a
- I would not recommend it (“add” confusion)

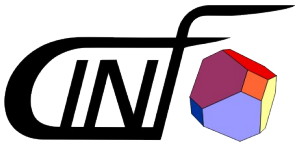


Center for Individual Nanoparticle Functionality

- Introduction
- Git data structures part 1
- **Git data structures part 2**
- Branches
- Network structure
- Summary

# svn to git
# 15+ years of new concepts

# The git data structure



- Three kinds of objects;
  - **Blobs**: File content
  - **Tree**: Directory manifest
    - why oh why, would they call those trees
  - **Commit**: A saved snapshot
- (almost always) clear text, hashed and zipped
- Each object is saved in a file
- The filename **is the hash**

# Each snapshot is a directory tree

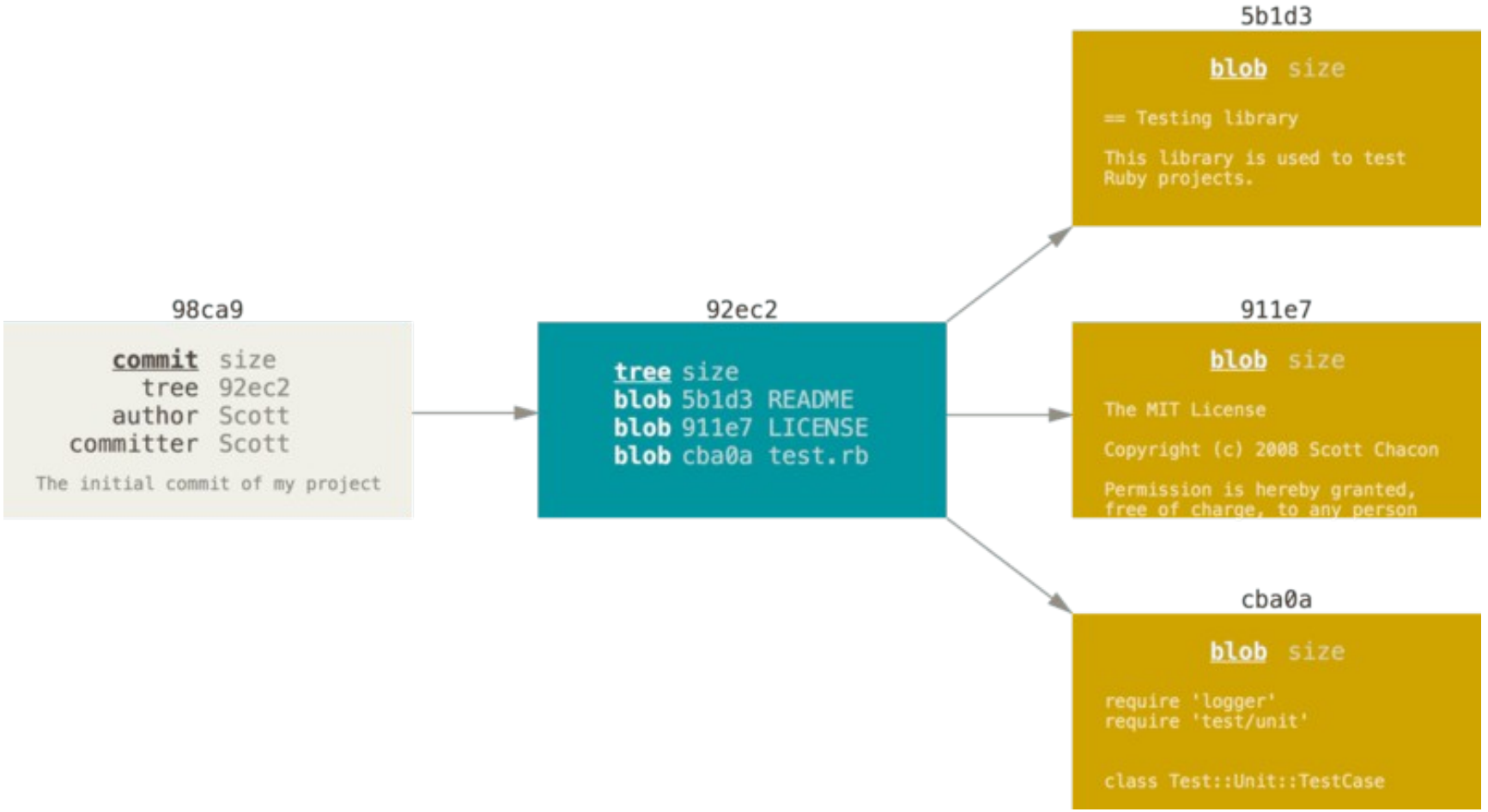



Figure from “Pro Git” by Scott Chacon, https://git-scm.com/book/en/v2

# Commits reference their parents

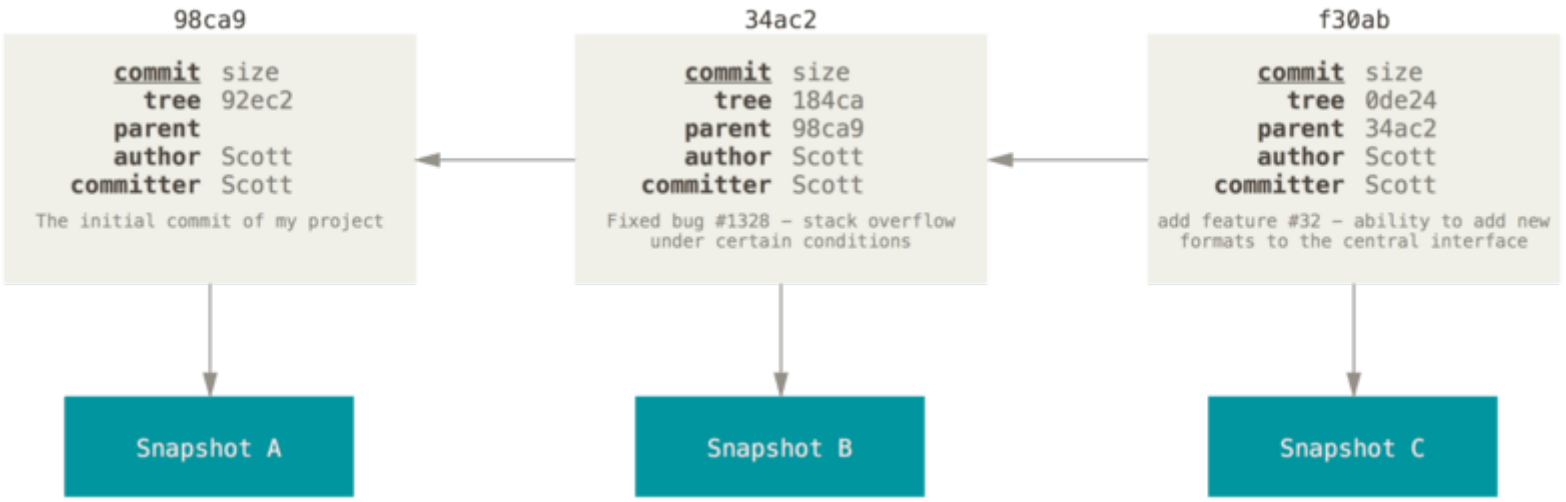
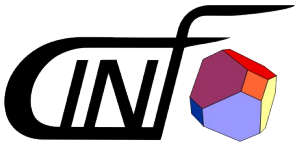



Figure from “Pro Git” by Scott Chacon, https://git-scm.com/book/en/v2

# Explains a few things

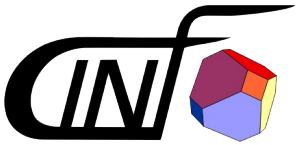


- Neat, almost simplistic design
- git is all about clever algorithms to do usefull stuff around a simple data structure

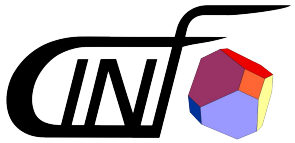
- “Snapshotting, doesn't that use a lot of space?”
- No, because git reuses all the objects it can

- (Eventually git will delta compress all the old objects into a pack file and gain even more space efficiency)

# Git checks its checksums



- Hashes are checked on checkout
- Prevents un-detected repository corruption or tampering
- If you have the hash, you are guarantied to get the same bits back



Center for Individual Nanoparticle Functionality

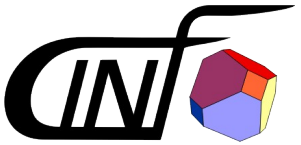
- Introduction
- Network structure
- Git data structures part 1
- Git data structures part 2
- **Branches**
- Summary

# svn to git
# 15+ years of new concepts

# Branches

A branch, in git, is nothing more or less than
**a pointer into the tree of commits**

# Branches



- Branches are cheap
  - A 41 byte file (really)
- Only you ever see them
  - Remember that you have your own copy on Gitlab to work up against
- Git is good are merging them
- Branches are good for containment of work
- … and in git they are usable
- All work should take place in a feature branch

# A single branch

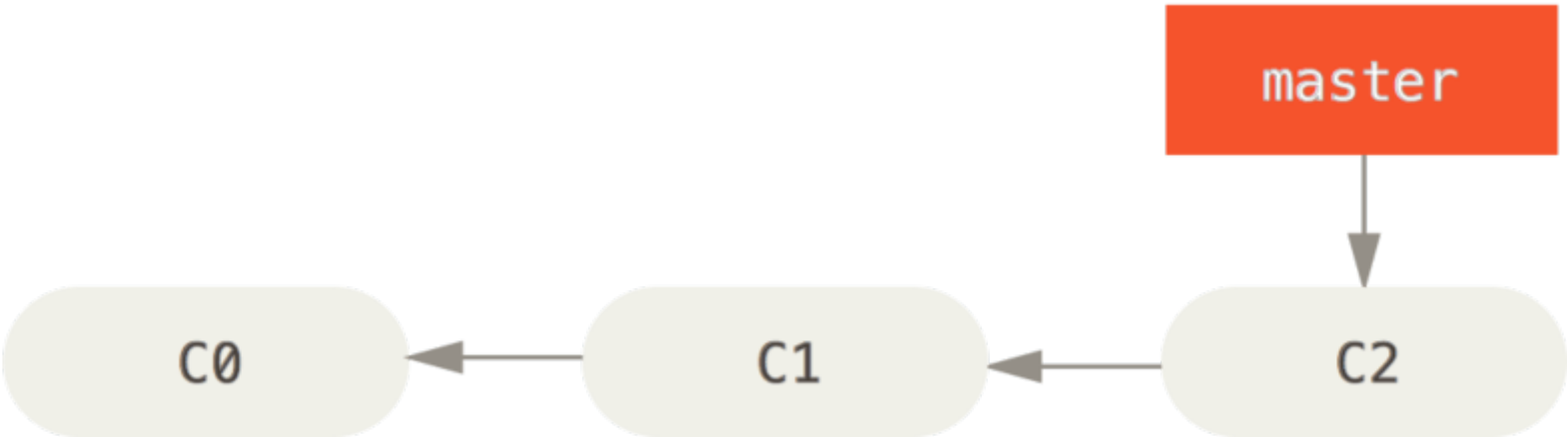



Figure from “Pro Git” by Scott Chacon, https://git-scm.com/book/en/v2

# Branching (adding a branch)

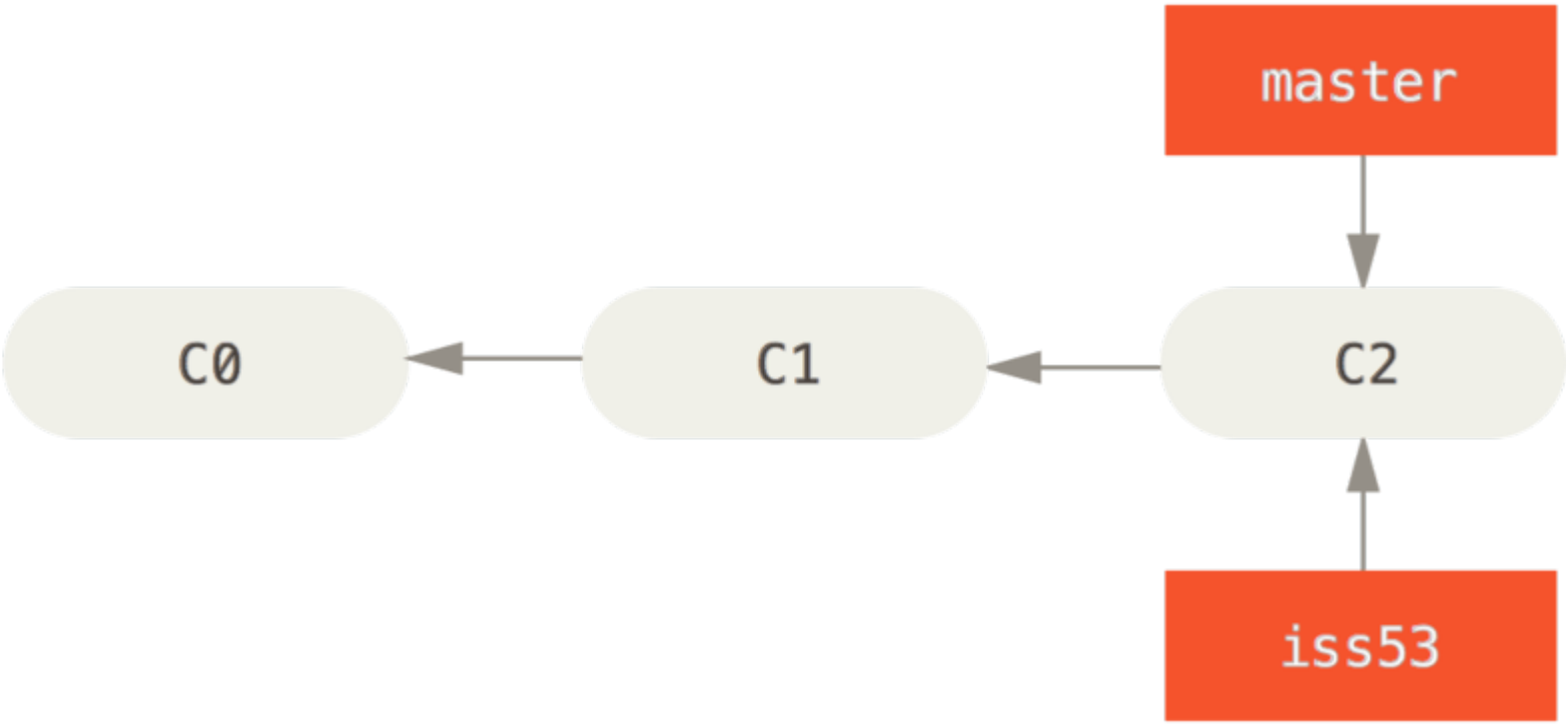
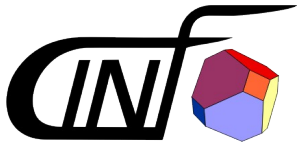



Figure from “Pro Git” by Scott Chacon, https://git-scm.com/book/en/v2



# DEMO

# Advancing the branch

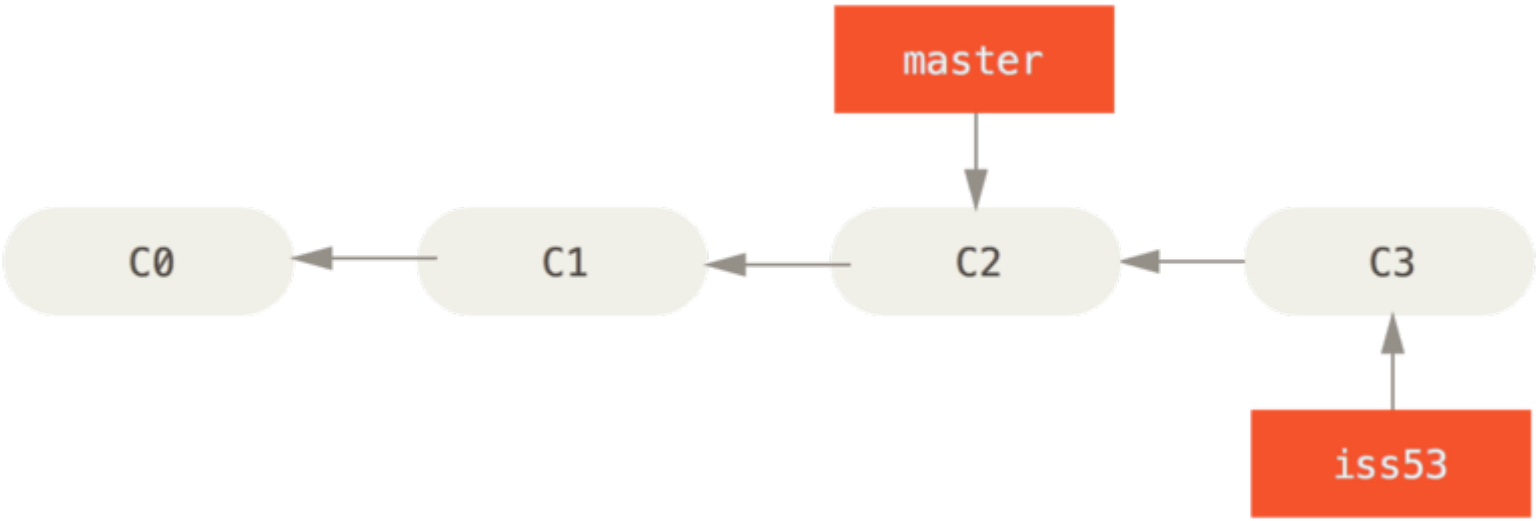



Figure from “Pro Git” by Scott Chacon, https://git-scm.com/book/en/v2

# Adding another branch

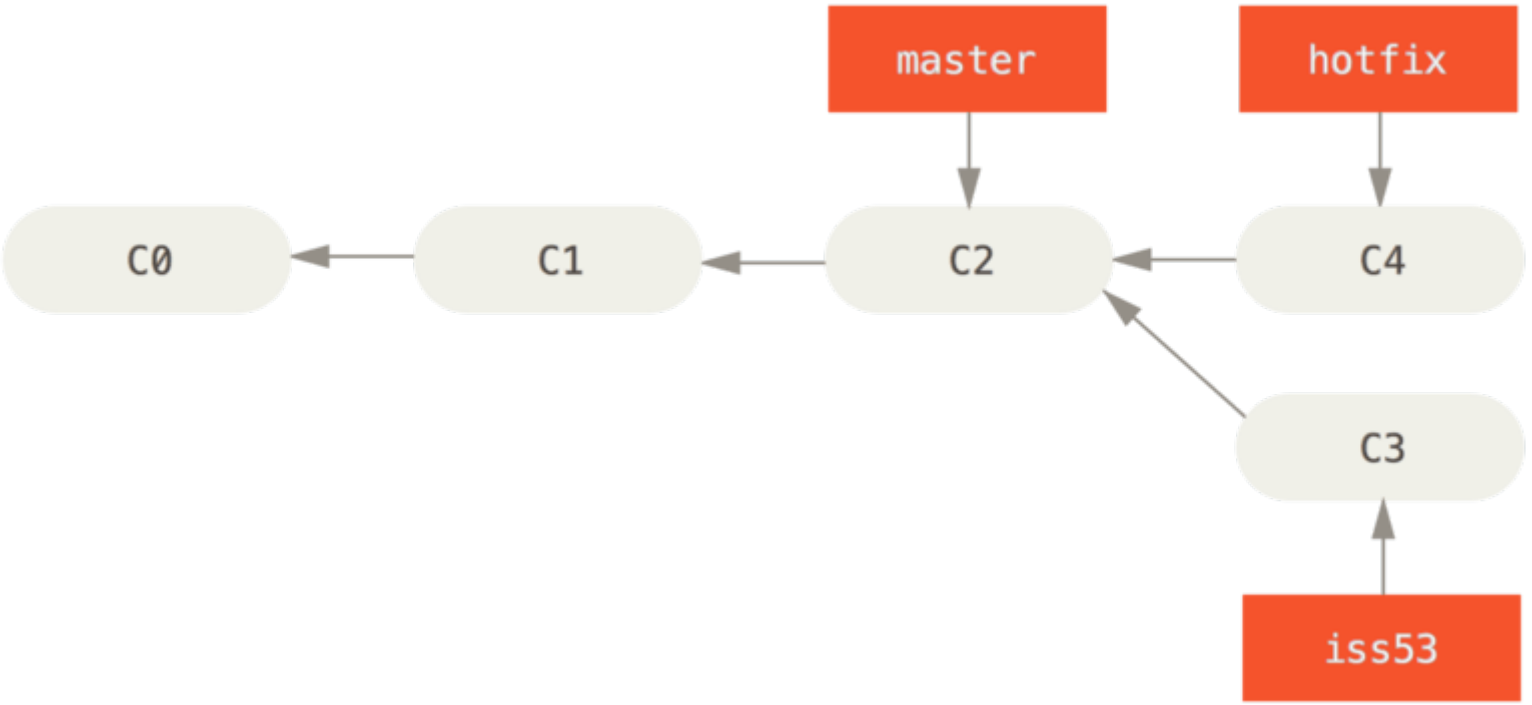



Figure from “Pro Git” by Scott Chacon, https://git-scm.com/book/en/v2

# Merging, **fast forward** merge

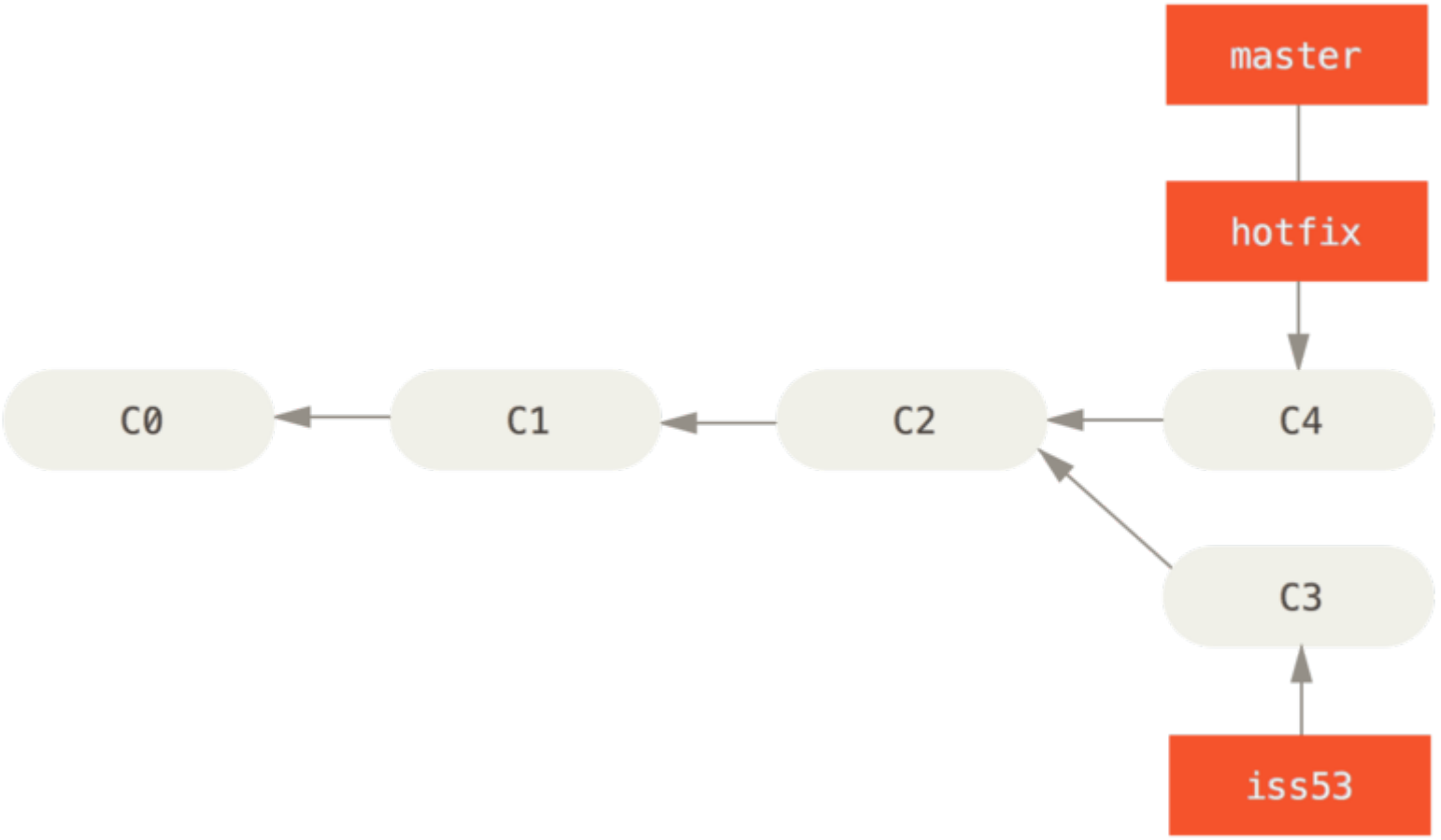



Figure from “Pro Git” by Scott Chacon, https://git-scm.com/book/en/v2

# Merging, **three way** merge

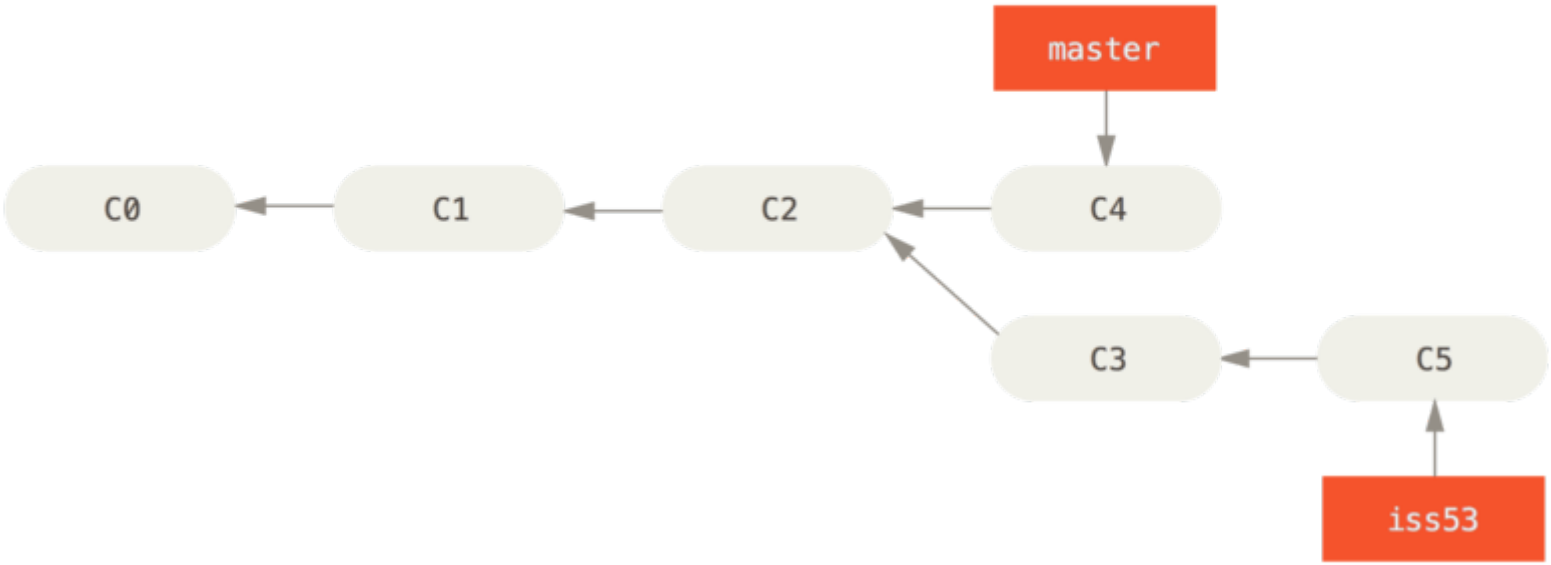



Figure from “Pro Git” by Scott Chacon, https://git-scm.com/book/en/v2

# Merging, **three way** merge

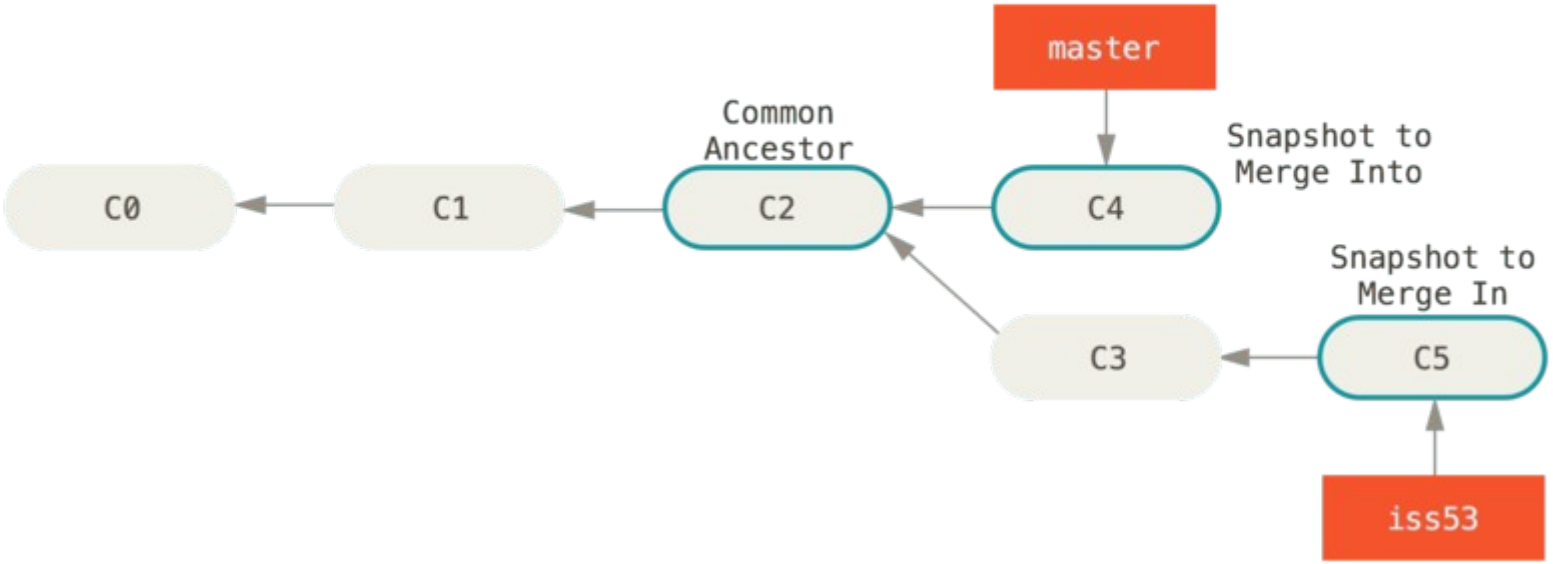



Figure from “Pro Git” by Scott Chacon, https://git-scm.com/book/en/v2

# Merging, **three way** merge

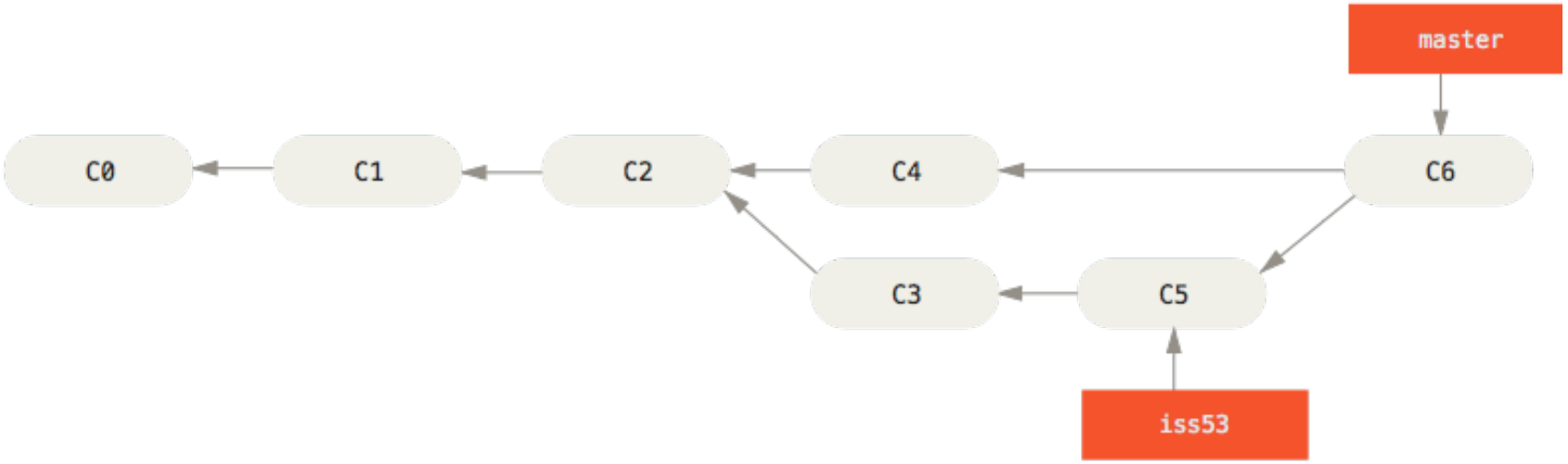
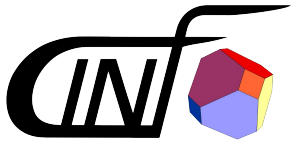



Figure from “Pro Git” by Scott Chacon, https://git-scm.com/book/en/v2

# Branches

- Use branches for everything
- Feature work (each on its own branch)
  - A new feature
  - A bug fix
  - An experiment
  - A savepoint, before doing something “dangerous”



Center for Individual Nanoparticle Functionality

- Introduction
- Git data structures part 1
- Git data structures part 2
- Branches
- **Network structure**
- Summary

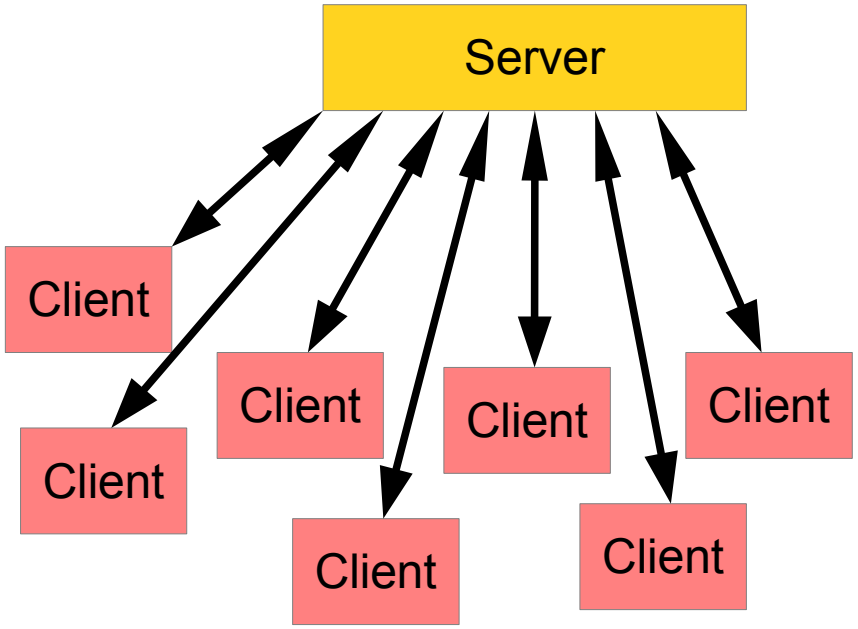
# svn to git
# 15+ years of new concepts

# Network structure

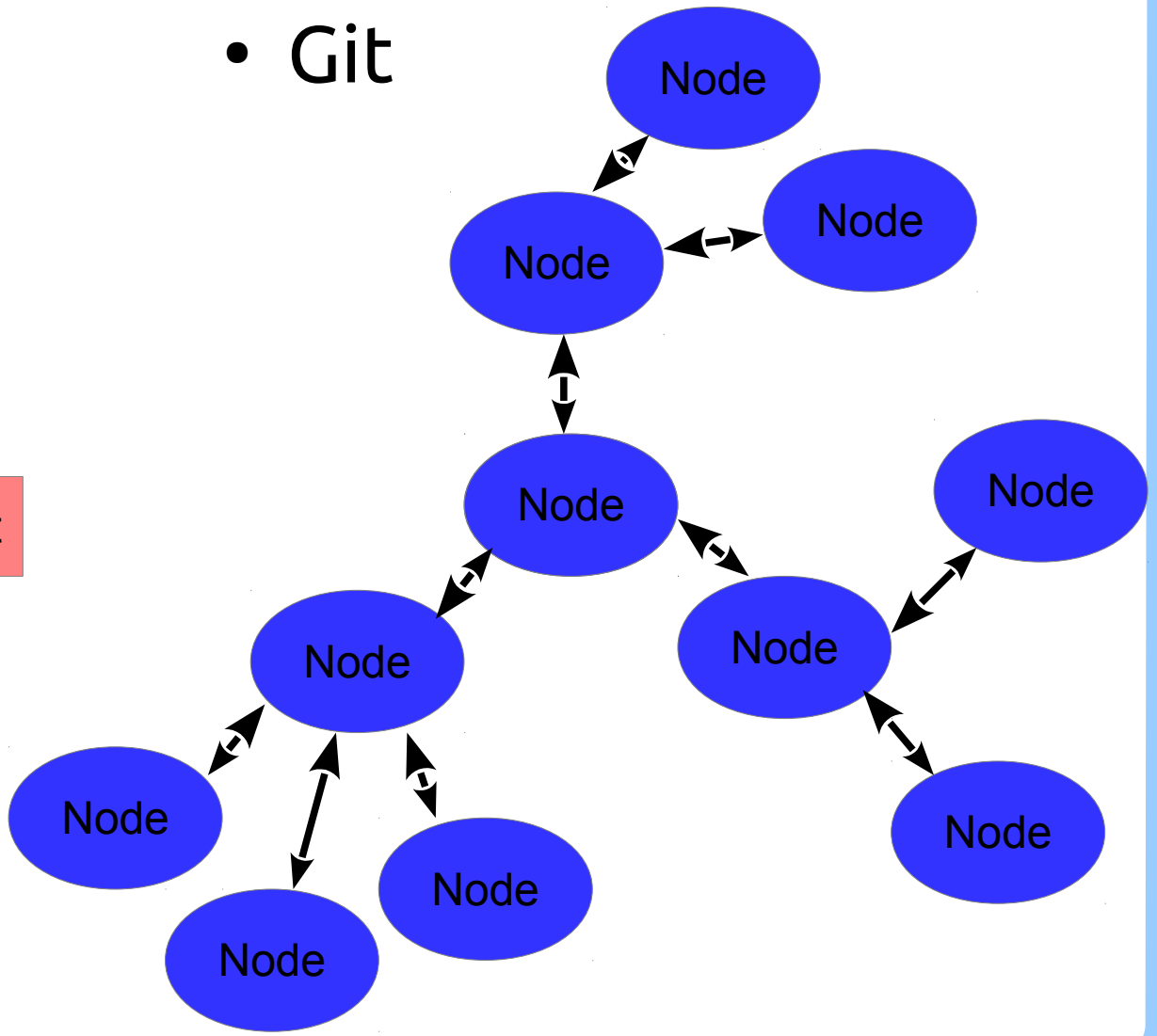
- Structure of “machines” involved with work on the project

# Network structure

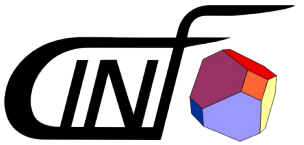
- SVN




- Git

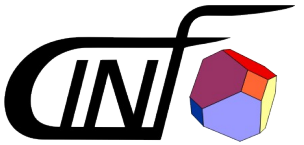


# Network structure in git



- Truly distributed
- No special one-true-server
- All nodes are the same
- Large flexibility
  - Maybe more than you need
- Git networks the same way trust does

# clone

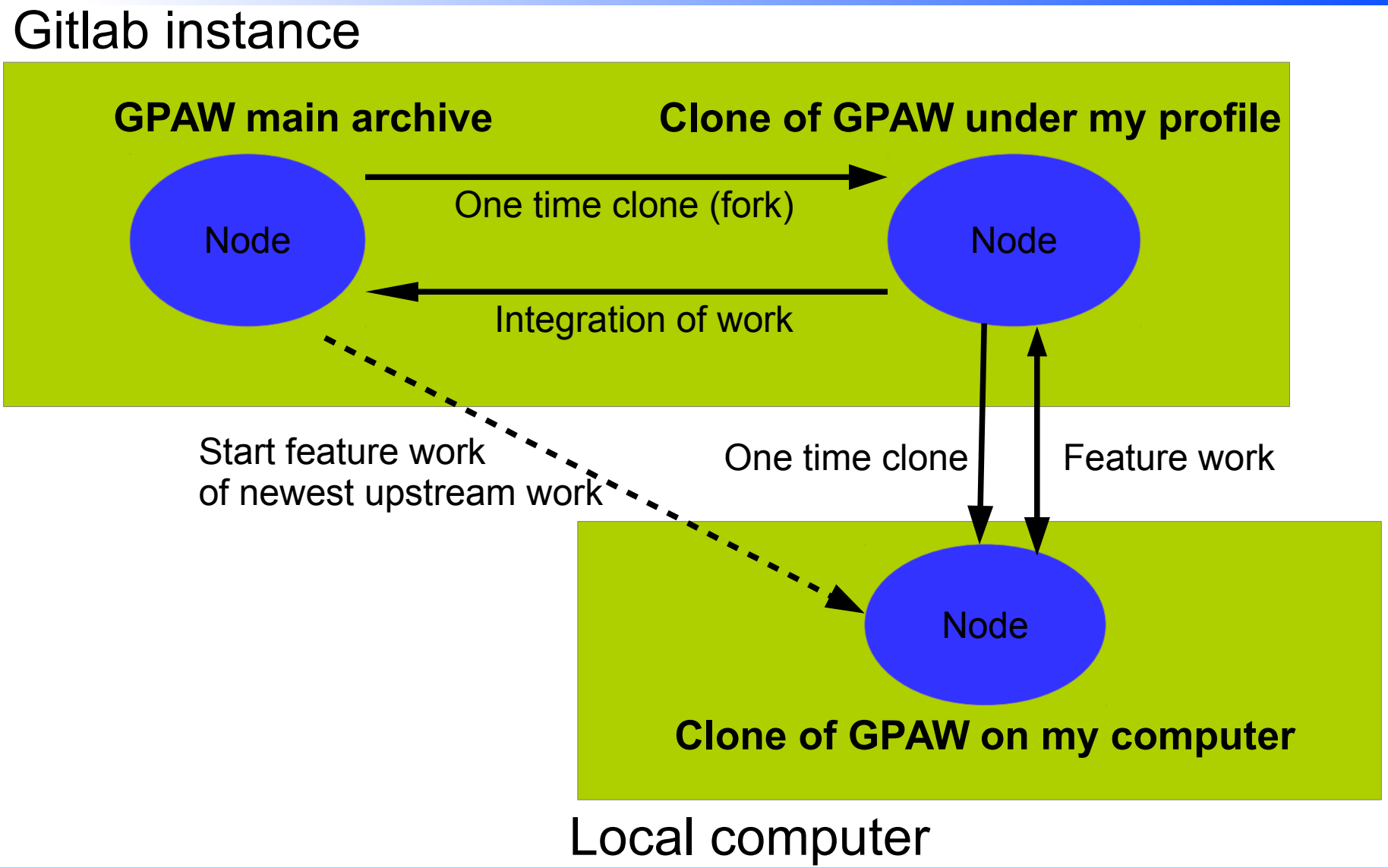


- `clone`, create a clone of an archive
- Clone means clone
  - You get it all
- The most common way to get an archive
- `git clone address_to_archive`

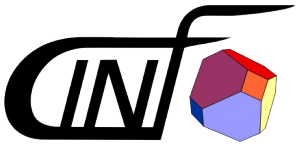
# remote

- A remote is a reference to **another** copy of the same archive
- Need not be a network location
- Cloning will automatically create a remote called `origin`

# Gitlab network structure

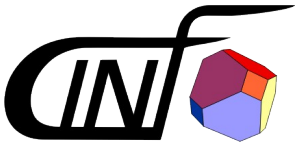


# Almost everything is local



- Have clone (all info)
- Most things are possible locally
- Git does as many things as it can locally
  - Fast, not limited by network speed
  - Available with no network

# Almost everything is local



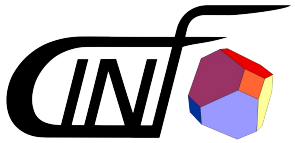
**Local**

- add
- commit
- branch
- merge
- log
- status

**Over the network**

- clone
- push
- fetch

- pull = fetch + merge



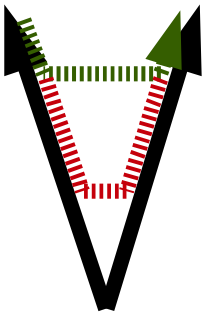
Center for Individual Nanoparticle Functionality

- Introduction
- Git data structures part 1
- Git data structures part 2
- Branches
- Network structure
- **Summary**

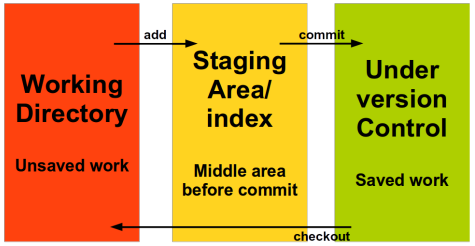
# svn to git
# 15+ years of new concepts

# Factlets, part 1

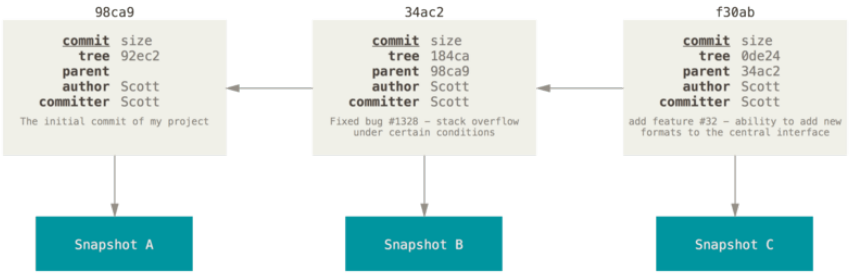
- git != SVN



- “add” adds to the staging area before commit




- gits history is a tree of snapshots

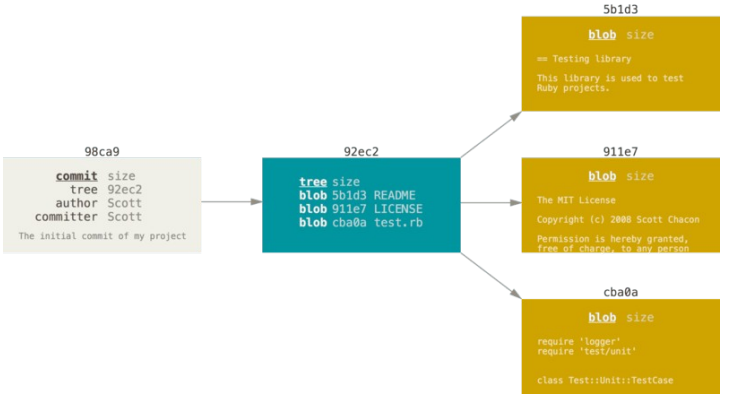





Figures from “Pro Git” by Scott Chacon, https://git-scm.com/book/en/v2

# Factlets, part 2

- gits stores data by hashes (and checks them)




- No node, in a network of similar repositories, is special (except by convention/placement)

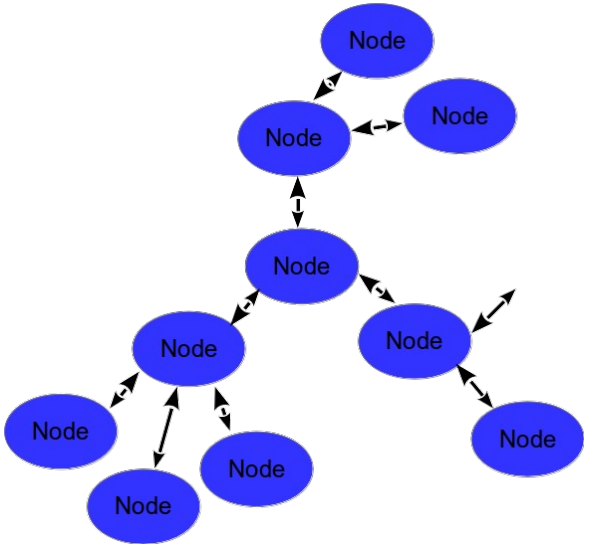
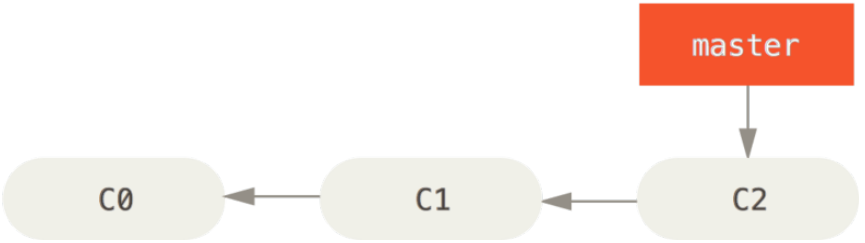





Figure from “Pro Git” by Scott Chacon, https://git-scm.com/book/en/v2

# Factlets, part 3

- branches are pointers into the tree of commits
  - they are cheap
  - they are local (untill explicitely pushed)




- … and git is good at merging them

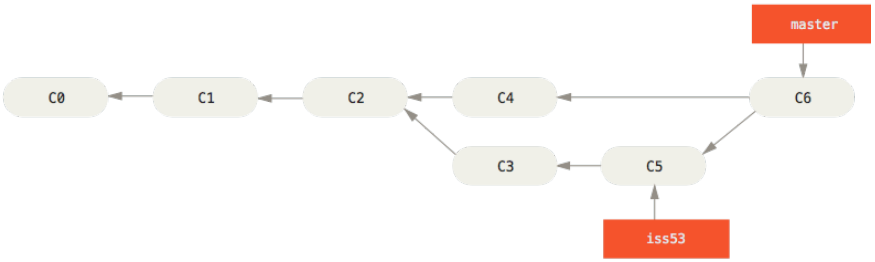





Figure from “Pro Git” by Scott Chacon, https://git-scm.com/book/en/v2

# Factlets, part 4

- merging into a branch, visible from the branch to be merged, is a fast forward merge
  - moves a pointer and **does not** require a commit

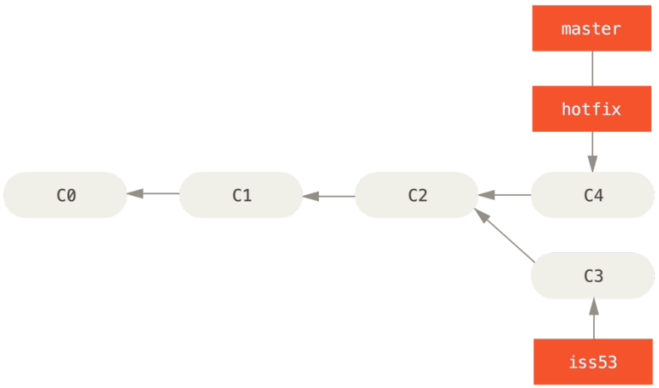





Figure from “Pro Git” by Scott Chacon, https://git-scm.com/book/en/v2

THE END

- Presentation at
- git clone https://github.com/KennethNielsen/presentations
- But really, watch the videos ;)