exact

Giuseppe Piero Brandino

CRS – OGS Udine – 26 Novembre 2019

Introduction to version control and Git

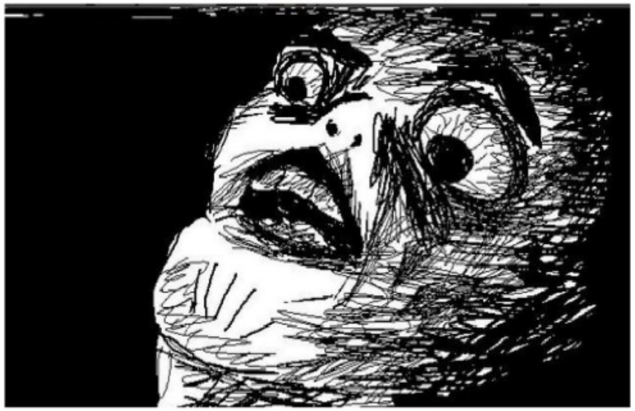
- The problems with lots of code and lots of people
- Version control systems
  - what are they?
  - how are they used?
  - centralized versus distributed version control
  - Features of version control including branching
  - Introduction to Git

Dealing with Change

- How do you manage your code regarding
    - Modifying existing code
    - Backing up working code
    - Checking if an idea works
    - Sharing code in group projects

(Bad) Solutions

- Copying (mycode_working.c, mycode_tmp.c)
- Copy & Paste code snippets
- Copy entire directories
- Emailing code to people (or to yourself...)

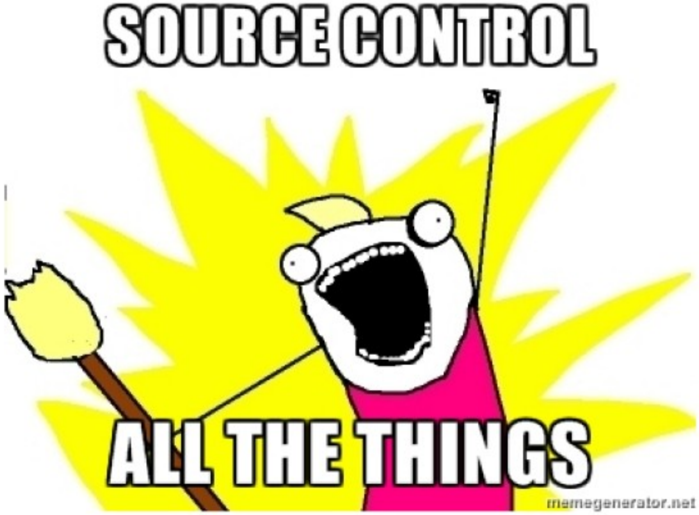


Making a mess – Managing the linux kernel

- 26 millions lines of code (end of 2018)
- The Linux kernel runs on different processors (ARM, x86, MIPS). These can require significant differences in low level parts of the code base
- Many different modules
- Old versions are required for legacy systems
- Because it is open source, any one can download and suggest changes.
- How can we create a single kernel from all of this?

Not just code!

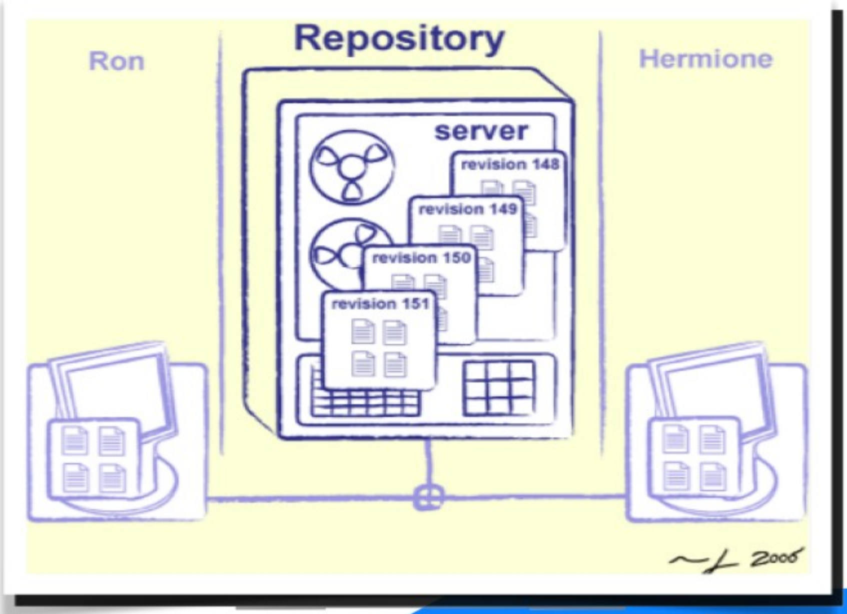
- A *Code Base* does not just mean code!
- Also includes:
  - Documentation
  - Build Tools (Makefiles etc)
  - Configuration files
- But NOT a certain type of file (executables, binaries)



Control the process automatically

- Manage these things using a version control system (VCS)

- A version control system is a system which allows for the management of a code base.



Details of the process

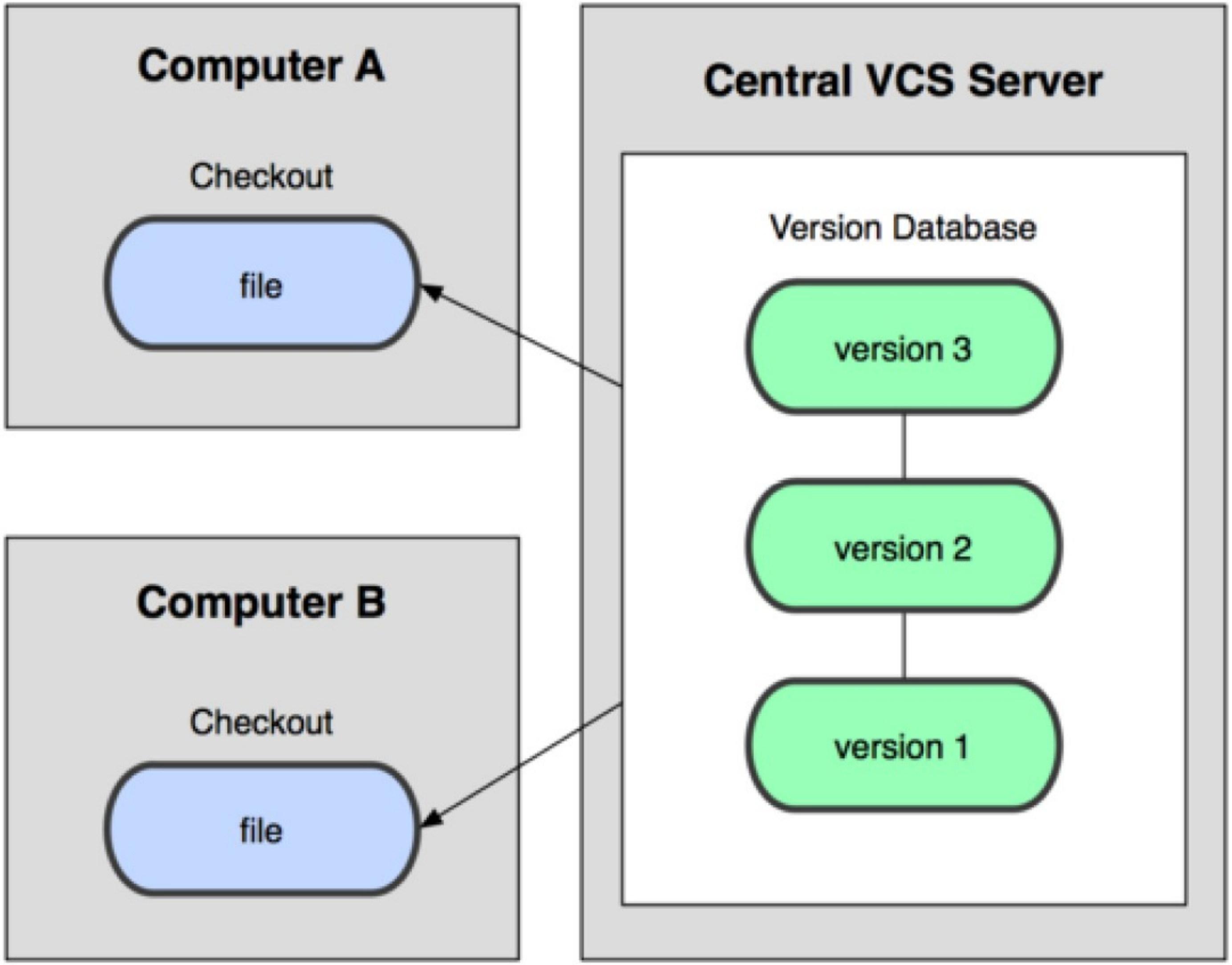
- Files are kept in a *repository*
- Repositories can be local or remote to the user
- The user edits a copy called the *working copy*
- Changes are *committed* to the repository when the user is finished making changes
- Other people can then access the repository to get the new code
- Can also be used to manage files when working across multiple computers

Centralised Version Control

- A single server holds the code base
- Clients access the server by means of check-in/check-outs
- Examples include CVS, Subversion, Visual Source Safe.

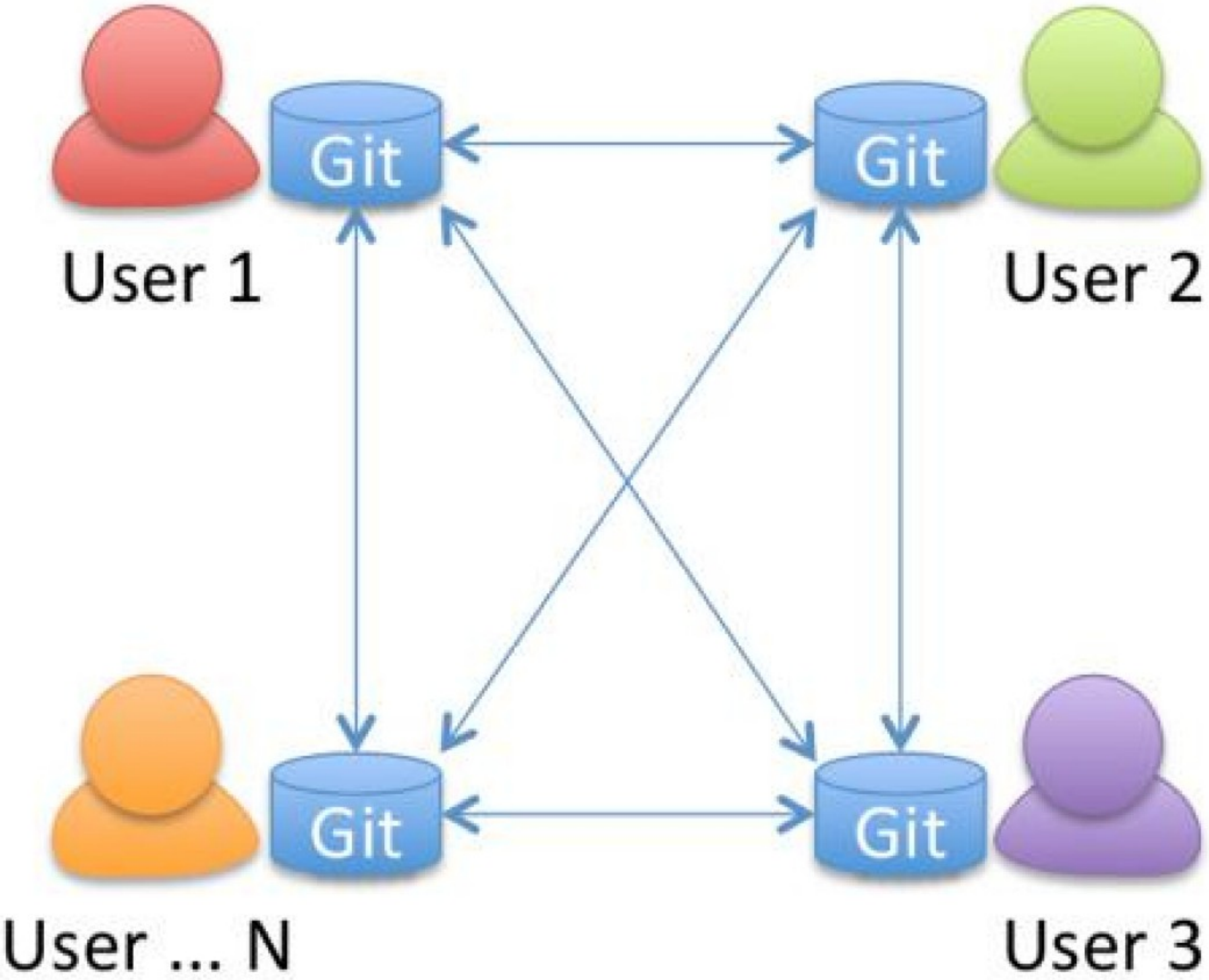
Advantages: Easier to maintain a single server.

Disadvantages: Single point of failure.


Computer A
Checkout
file
Central VCS Server
Version Database
version 3
version 2
version 1
Computer B
Checkout
file

Distributed Version Control

- Each client (essentially) holds a complete copy of the code base.
- Code is shared between clients by push/pulls
  - Advantages: Many operations cheaper. No single point of failure
  - Disadvantages: A bit more complicated!

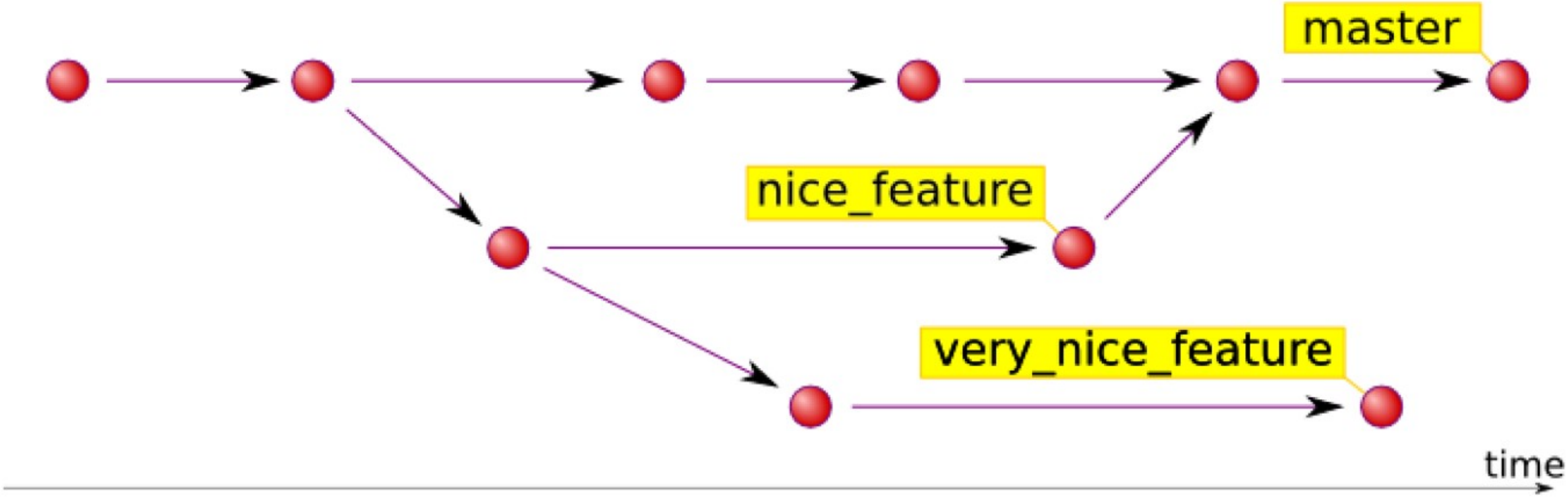

User 1
Git
Git
User 2
Git
Git
User ... N
User 3

More Uses of Version Control

- Version control is not just useful for collaborative working, essential for quality source code development
- Often want to undo changes to a file
  - start work, realize it's the wrong approach, want to get back to starting point
  - like "undo" in an editor…
  - keep the whole history of every file and a *changelog*
- Also want to be able to see who changed what, when
  - The best way to find out how something works is often to ask the person who wrote it

Branching

- Branches allows multiple copies of the code base within a single repository.
  - Different customers have different requirements
    - Customer A wants features A,B, C
    - Customer B wants features A & C but not B because his computer is old and it slows down too much.
    - Customer C wants only feature A due to costs
  - Each customer has their own branch.
- Different versions can easily be maintained


master
nice_feature
very_nice_feature
time

Merging

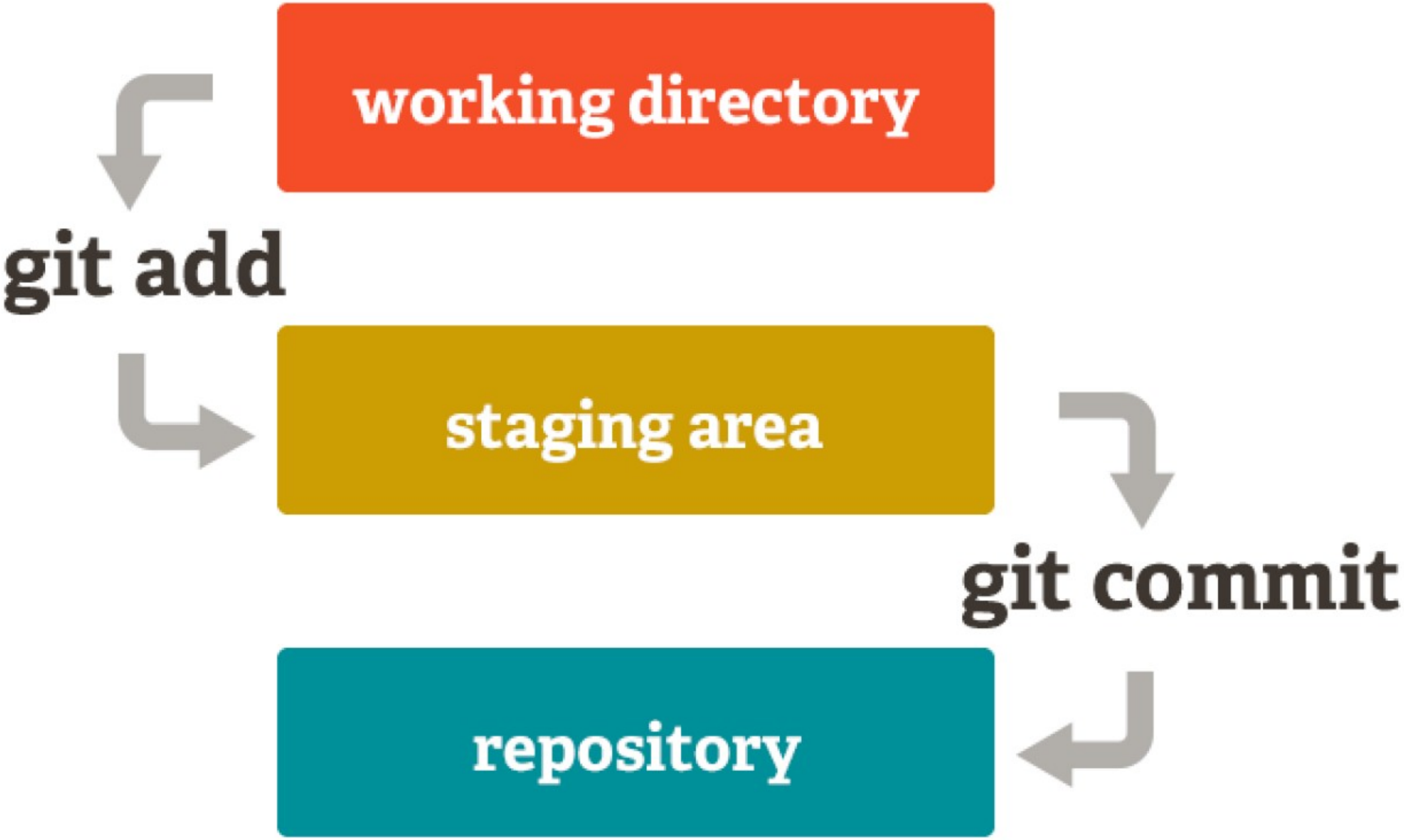
- There are occasions when multiple versions of a file need to be collapsed into a single version.
  - E.g. A feature from one branch is required in another
- This process is known as a merge.
- Difficult and dangerous to do in CVS
- Easy and cheap to do it git

- Git
  - Distributed version control system
  - Alternative to SVN, StarTeam
- Github
  - repository site
  - Bitbucket, Gitlab

Git basics

- https://git-scm.com/download
- https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

- Let’s define ourselves first
  - `git config --global user.email "you@example.com"`
  - `git config --global user.name "Your Name"`



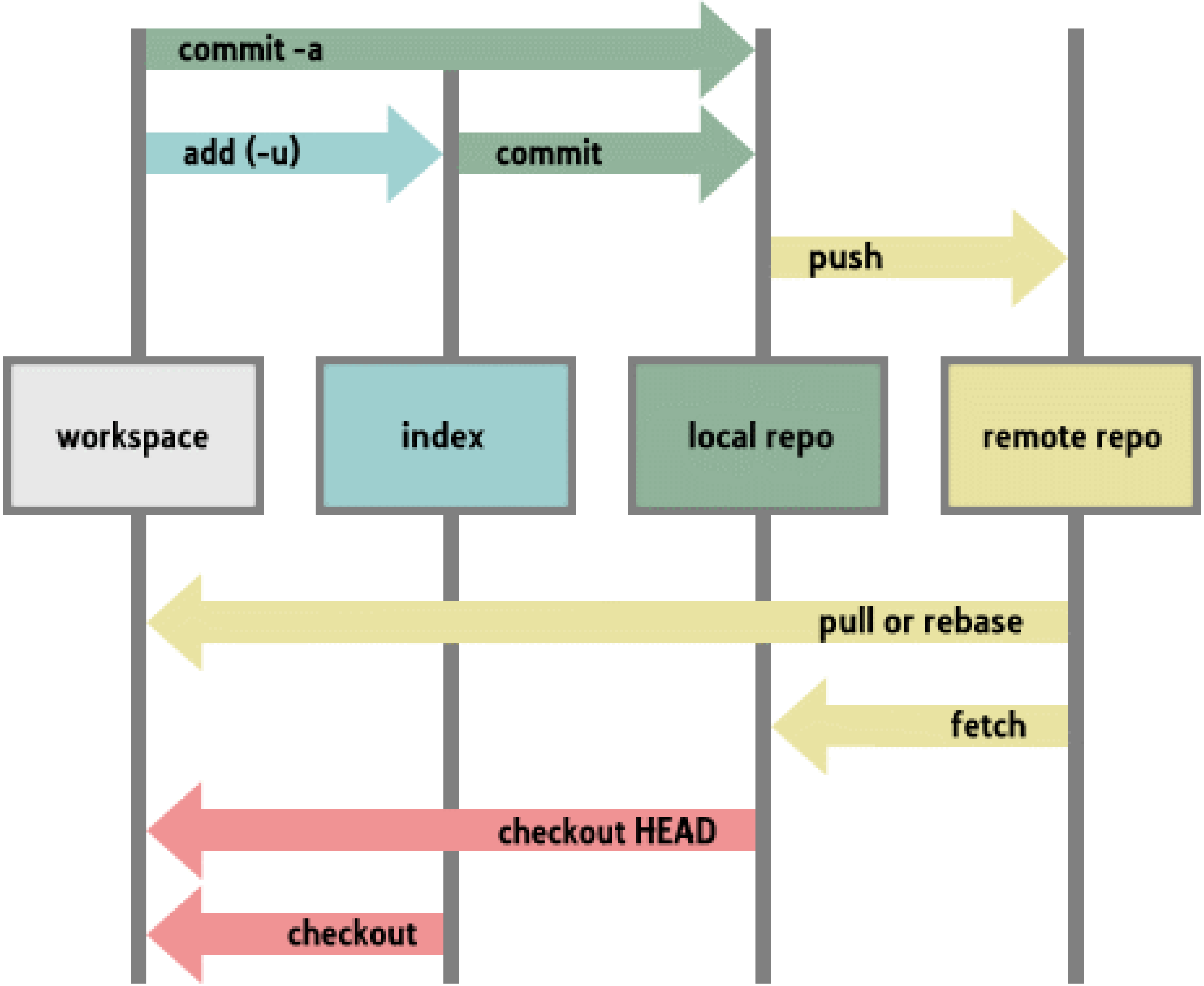
working directory

git add

staging area

git commit

repository


commit -a
add (-u)
commit
push
workspace
index
local repo
remote repo
pull or rebase
fetch
checkout HEAD
checkout

Local first

- first lets make a git folder in our computer
  - git init
- add a file to the folder
  - git add newfile.file
  - git status
  - git commit -m “new file added”

Ignore some file?

- Create a file called .gitignore
  - Write the file names that should be ignored by git
  - Commit the file

Then remote

- Create a repo on Github
- Add the remote
  - `git remote add origin <repo url>`
- Now you can push
  - `git push origin master`

Remove a file

- git rm somefile.txt
- git commit –m ‘removed’
- git push origin master

Create a branch

- git checkout -b development
- modify newfile.txt
- git add newfile.txt
- git commit –m ‘removed’
- git push origin development

Merge a branch

- git checkout master
- git merge development

Pulling a repository/editing

- lets clone a repository from github
    - git clone https://github.com/cosai/test

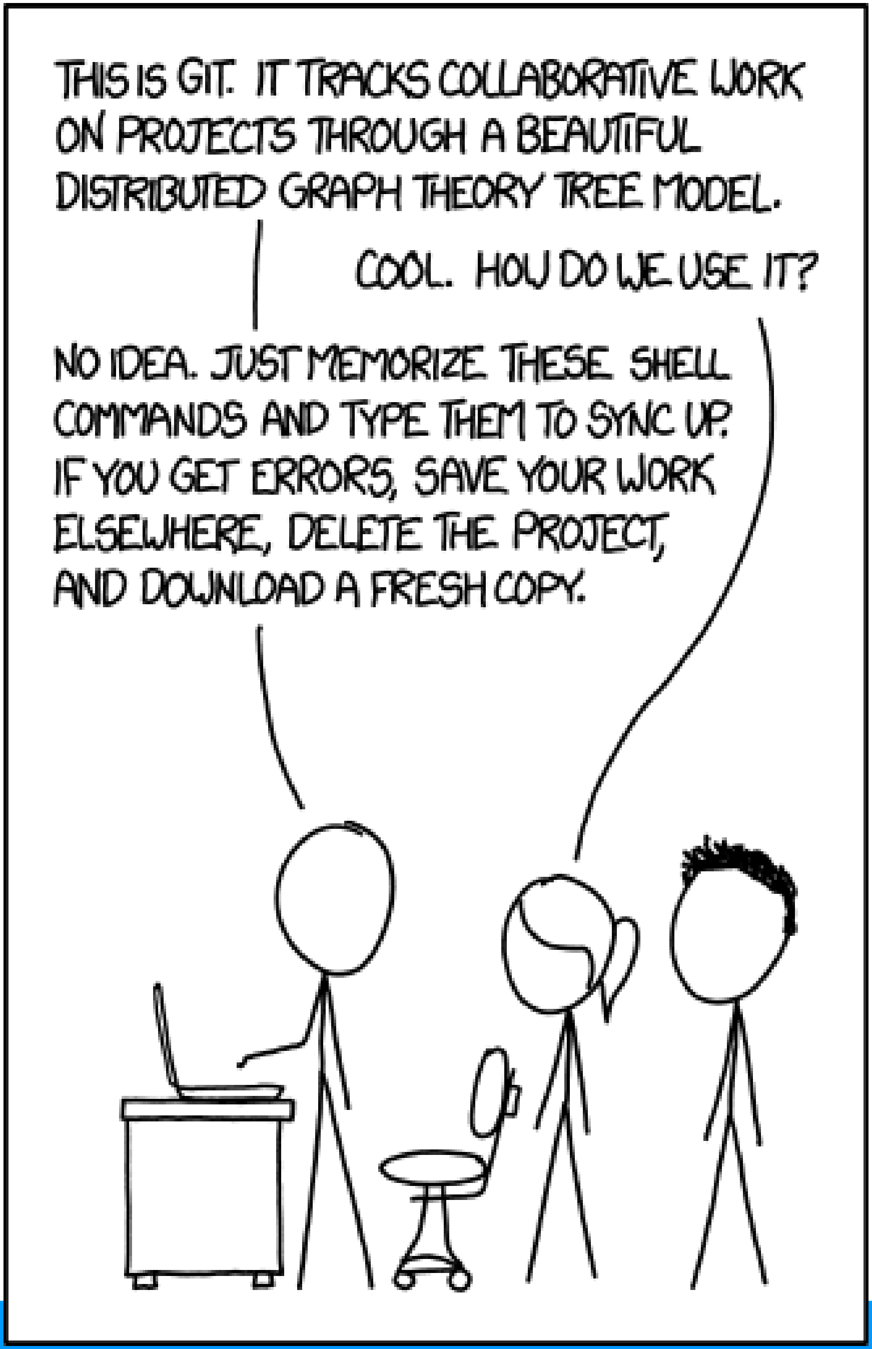
- Edit the file a.txt
- git add a.txt
- git status
- git commit –m ‘something added’
- git push origin

Going back

- Git log
  - Show me the logs
- See the commit id
  - git reset --hard HEAD
- Destroys the local modifications!
- git clean
- Removes untracked files!

One step back!

- An easy way to revert last commit (1)
  - git revert HEAD~1
  - git push origin
- Use it with caution!
- Try not to rewrite history (if can avoid it)


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.