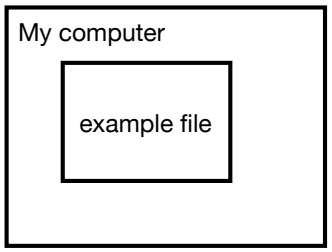
Home-baked Version Control ("low" tech)



This is the system in which you prefix your file names

me

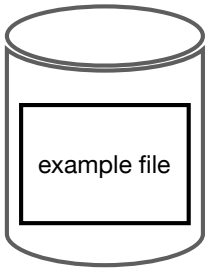
pros

lightweight
easy to establish
no merge conflicts
user has full control

cons

users are bad at consistency
humans are forgetful
accidental overwriting
All on one computer/ one location
scalability

Autocratic Version Control



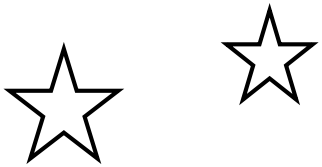
Remote repository

pros

access from remote
no merge conflicts

cons

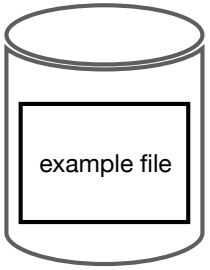
only one person has write access at a time



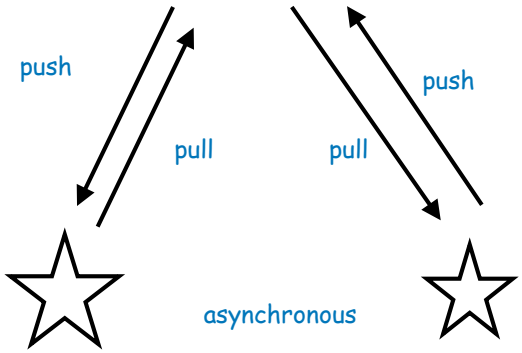
Users

Centralized Version Control (subversion, svn, perforce)

one master remote repository



Remote repository



asynchronous

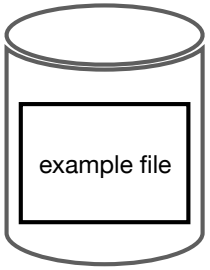
pros

can checkout a subtree of the repository

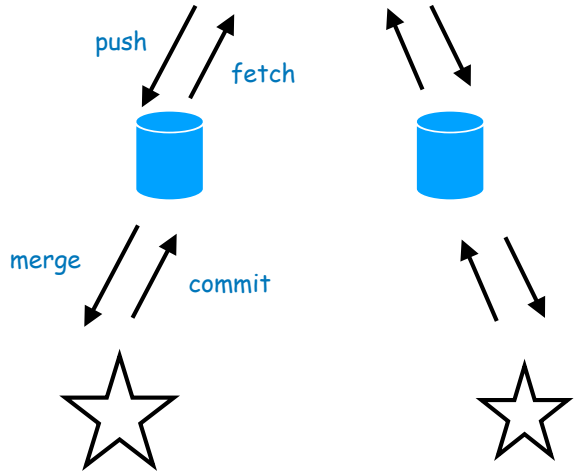
cons

Distributed Version Control (git, mercurial, bazaar)

Users also maintain their own local repositories



Remote repository

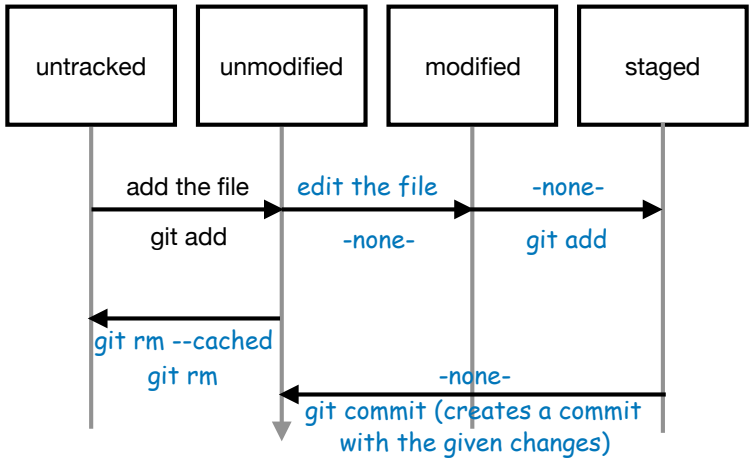


pros

all users have all parts of the repository

cons

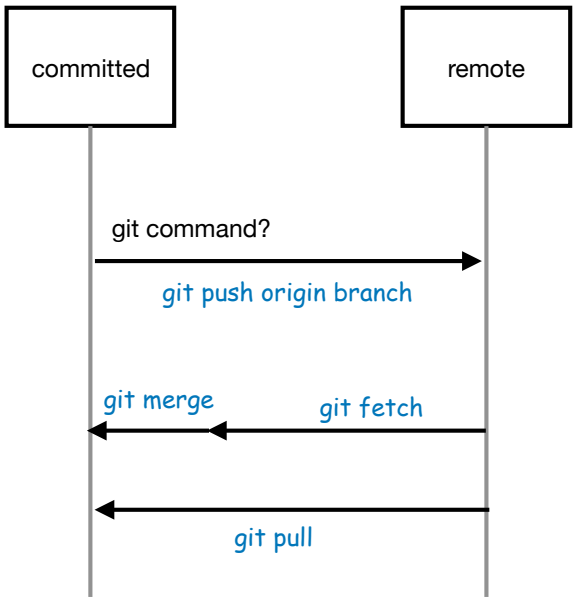
merge conflicts
branching logic can become extremely complicated
requires users to have full copies of the repositories locally (bigger)

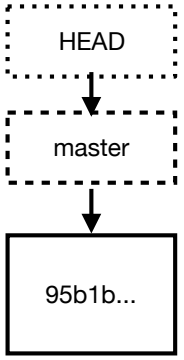


What is the state of the files after commit?

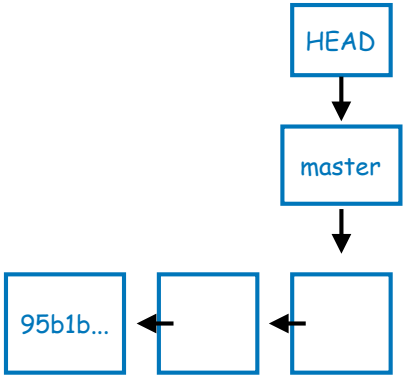
files are indexed and a checkpoint has been created in the log (they are still local though)

Label these arrows are follows: how? → git command?

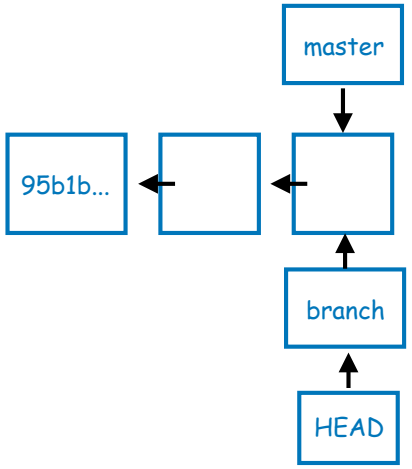




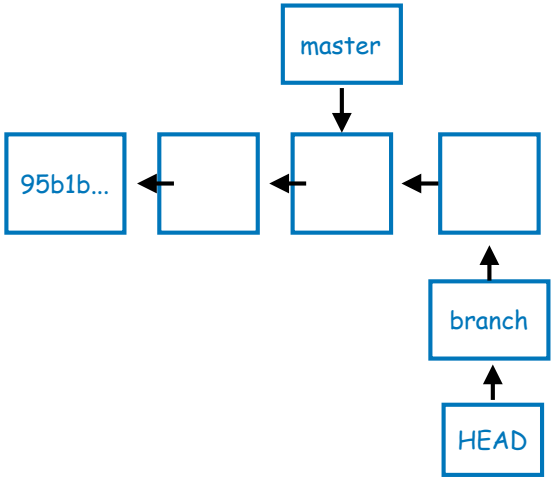
1) label each of the above boxes with what they are



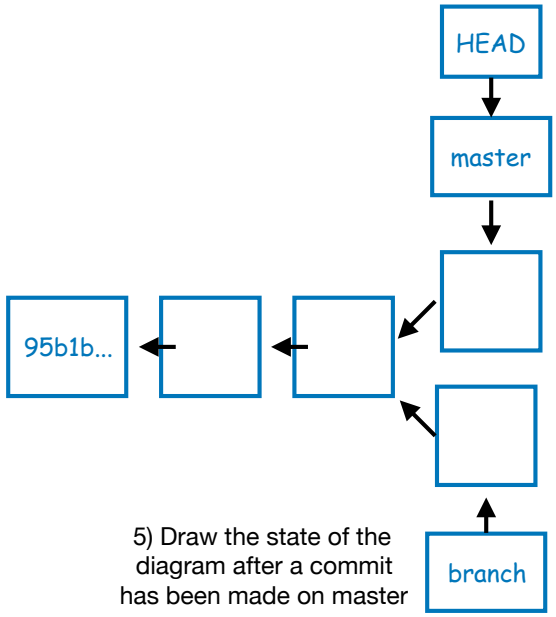
2) Draw the state of the diagram after 2 more commits have been made



3) Draw the state of the diagram after a branch has been created



4) Draw the state of the diagram after a commit has been made on our new branch



5) Draw the state of the diagram after a commit has been made on master