Change Management

Week 9

Cite Tomer Weiss

What Changes Are We Managing?

Software

- Planned software development
  - team members constantly add new code
- (Un)expected problems
  - bug fixes
- Enhancements
  - Make code more efficient (memory, execution time)

“The only constant in software development is change”

Q: What kind of information you want to keep for changes?

Features Required to Manage Change

- Backups
- Timestamps
- Who made the change?
- Where was the change made?
- A way to communicate changes with team

How to achieve that

- Figure out which parts changed (diff?)
- Communicate changes with team (patch?)
- But diff and patch are not that good

Disadvantages of diff & patch

- Diff requires keeping a copy of old file before changes
- Work with only 2 versions of a file (old & new)
  - Projects will likely be updated more than once
  - store versions of the file to see how it evolved over time

    index.html
    index-2009-04-08.html
    index-2009-06-06.html
    index-2009-08-10.html
    index-2009-11-04.html
    index-2010-01-23.html
    index-2010-09-21.html
- Numbering scheme becomes more complicated if we need to store two versions for the same date

Disadvantages of diff & patch

- Two people may edit the same file on the same date
  - 2 patches need to be sent and merged
- Changes to one file might affect other files (.h & .c)
  - Need to make sure those versions are stored together as a group

Version Control System

- Track changes to code and other files related to software
  - What new files were added?
  - What changes made to files?
  - Which version had what changes?
  - Which user made the changes?
  - Revert to previous version
- Track entire history of software
- Source control software
  - Git, Subversion (SVN), CVS, and others

Version Control System Allows you to

- See who introduced an issue and when
- see who last modified something that might be causing a problem
- compare changes over time
- revert the entire project back to a previous state
- revert files back to a previous state
- And more ...

Version Control System

- Three types
  - Local version control system
  - Centralized Version Control System
  - Distributed Version Control System

Local Version Control Systems

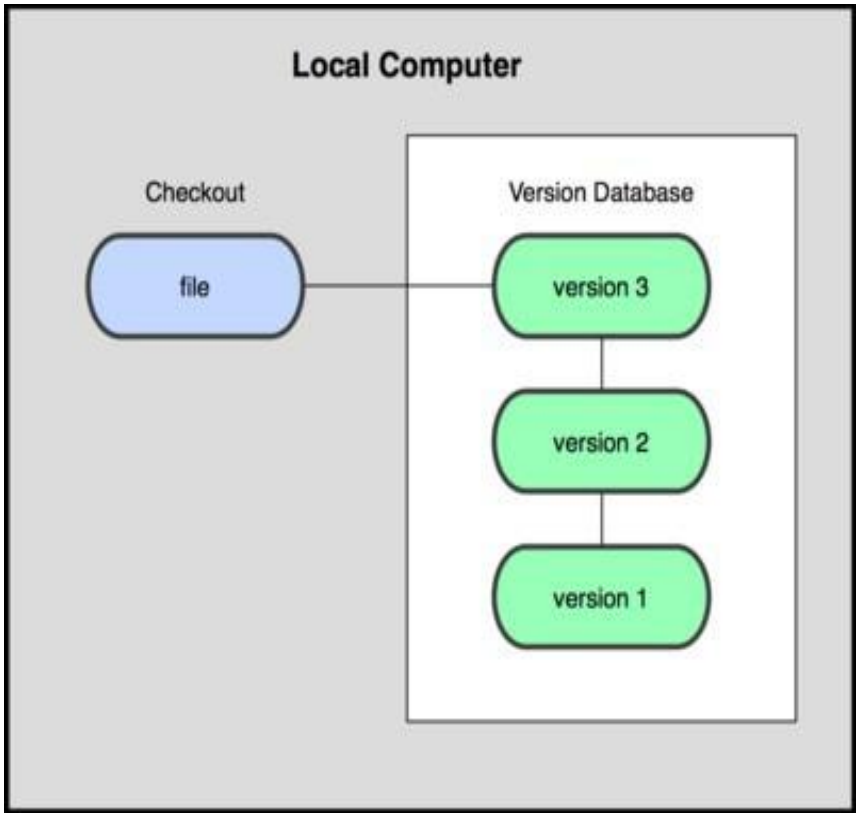


Image Source: git-scm.com

- Organize different versions as folders on the local machine
- No server involved
- Other users copy with disk/network
- Ex: rcs

Centralized Version Control System

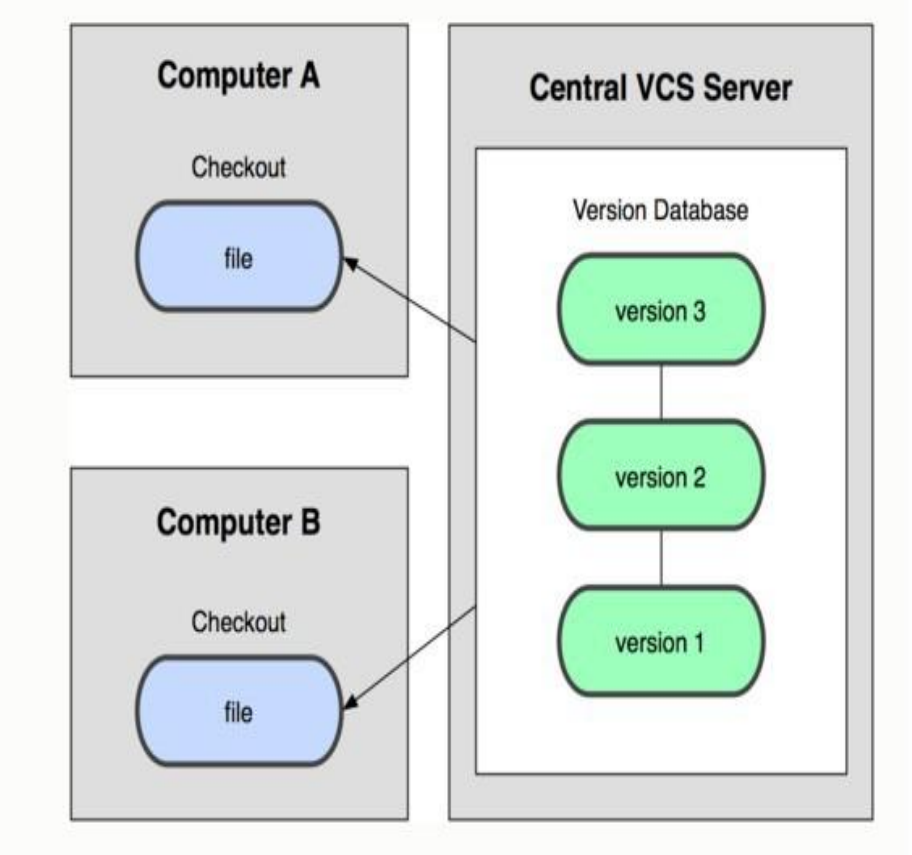


Image Source: git-scm.com

- Version history sits on a central server
- Users will get a working copy of the files
- Changes have to be committed to the server
- All users can get the changes
- Ex: Subversion (SVN)

Centralized: Pros and Cons

"The full project history is only stored in one central place."

Pros

- Everyone can see changes at the same time
- Simple to design

Cons

- Single point of failure (no backups!)

Distributed Version Control System

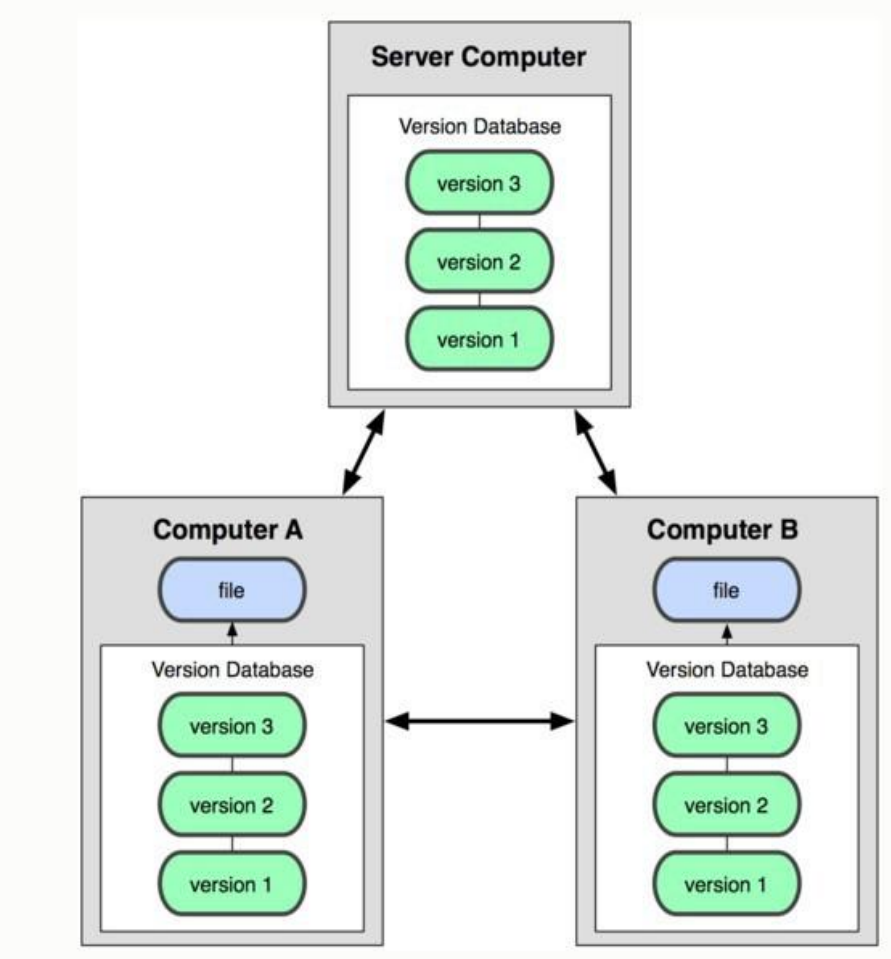


Image Source: git-scm.com

- Version history is replicated on every user's machine
- Users have version control all the time
- Changes can be communicated between users
- Ex: Git

Distributed: Pros and Cons

"The entire project history is downloaded to the hard drive"

Pros

- Commit changes/revert to an old version while offline
- Commands run extremely fast because tool accesses the hard drive and not a remote server
- Share changes with a few people before showing changes to everyone

Cons

- long time to download
- A lot of disk space to store all versions

Centralized vs. Distributed VCS

- Single central copy of the project history on a server
- Changes are uploaded to the server
- Other programmers can get changes from the server
- Examples: **SVN**, CVS

- Each developer gets the full history of a project on their own hard drive
- Developers can communicate changes between each other without going through a central server
- Examples: **Git**, Mercurial, Bazaar, Bitkeeper

SVN vs Git

- Git uses distributed version control
- Svn uses centralized version control

- Git uses a stream of snapshots to store data
- Svn uses files and a list of file-based changes

Data in SVN

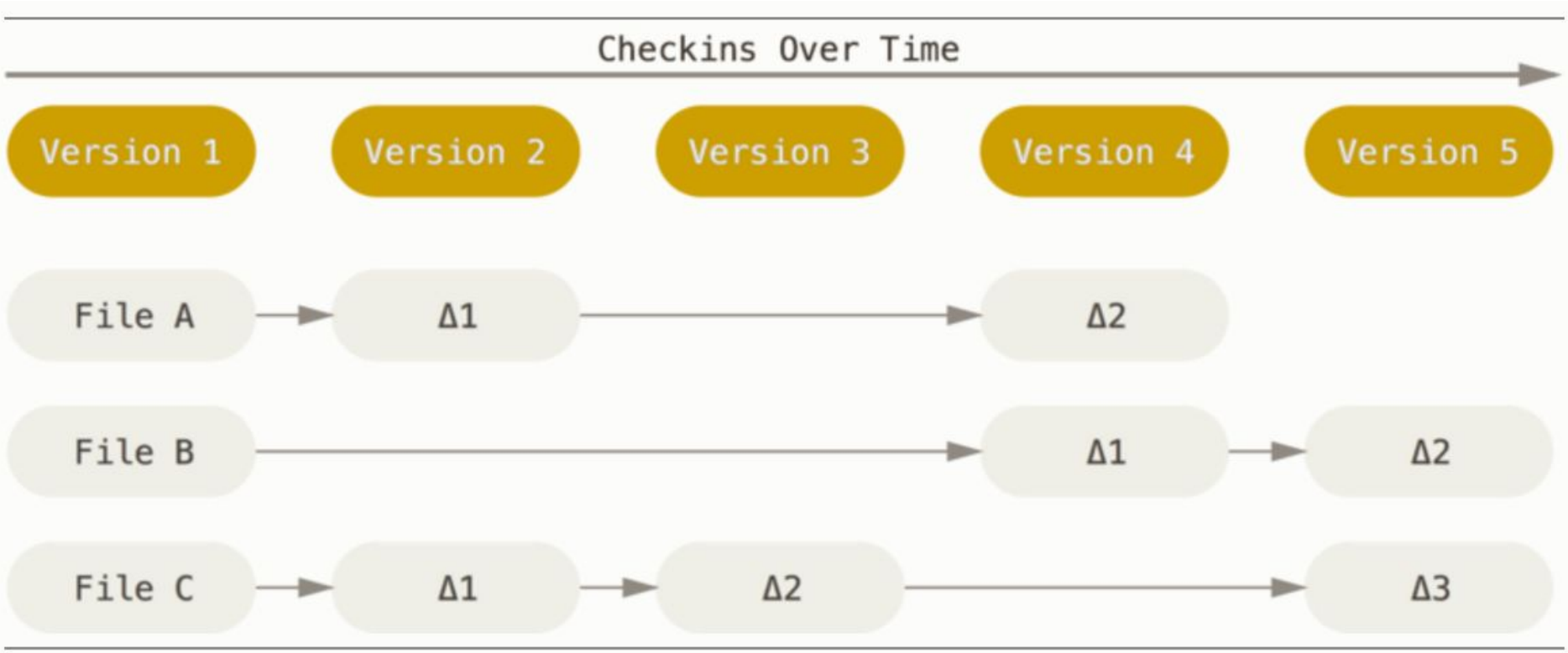


Figure 1-4. Storing data as changes to a base version of each file.

Cite: https://git-scm.com/book/en/v2/Getting-Started-Git-Basics

Data in Git

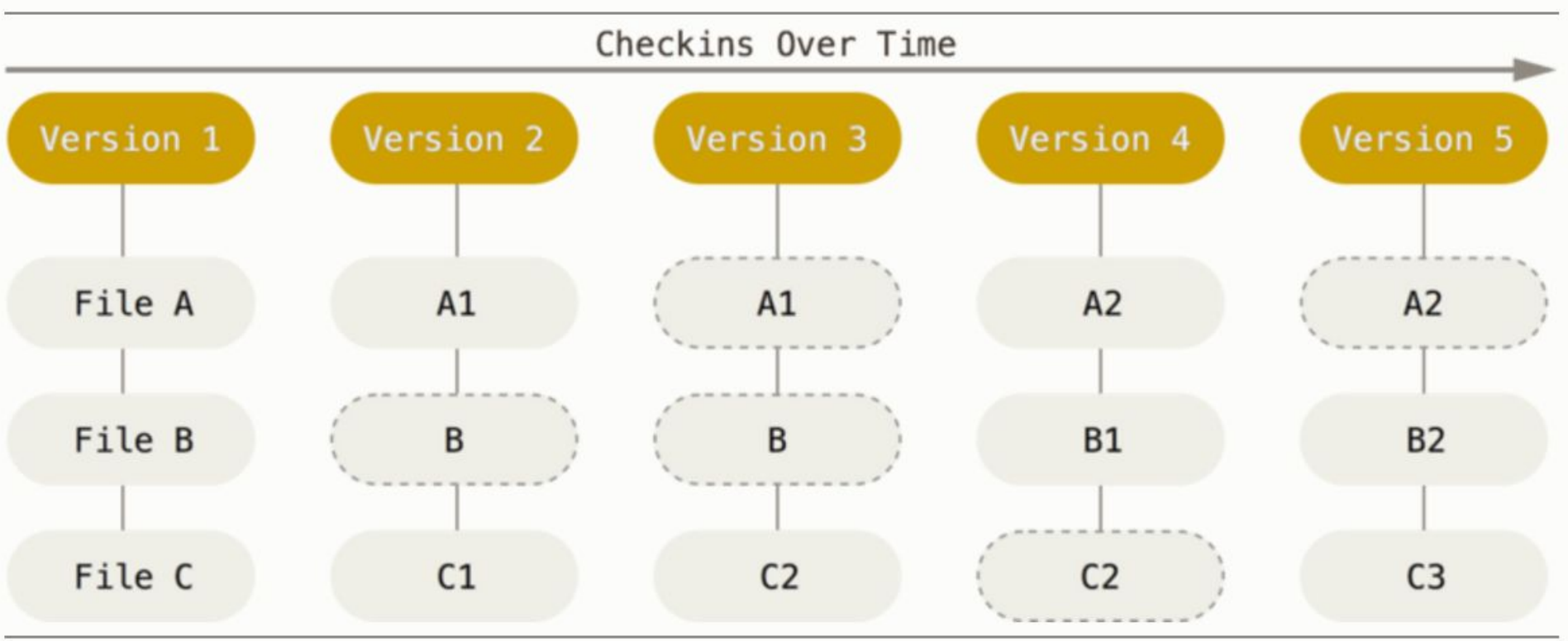


Figure 1-5. Storing data as snapshots of the project over time.

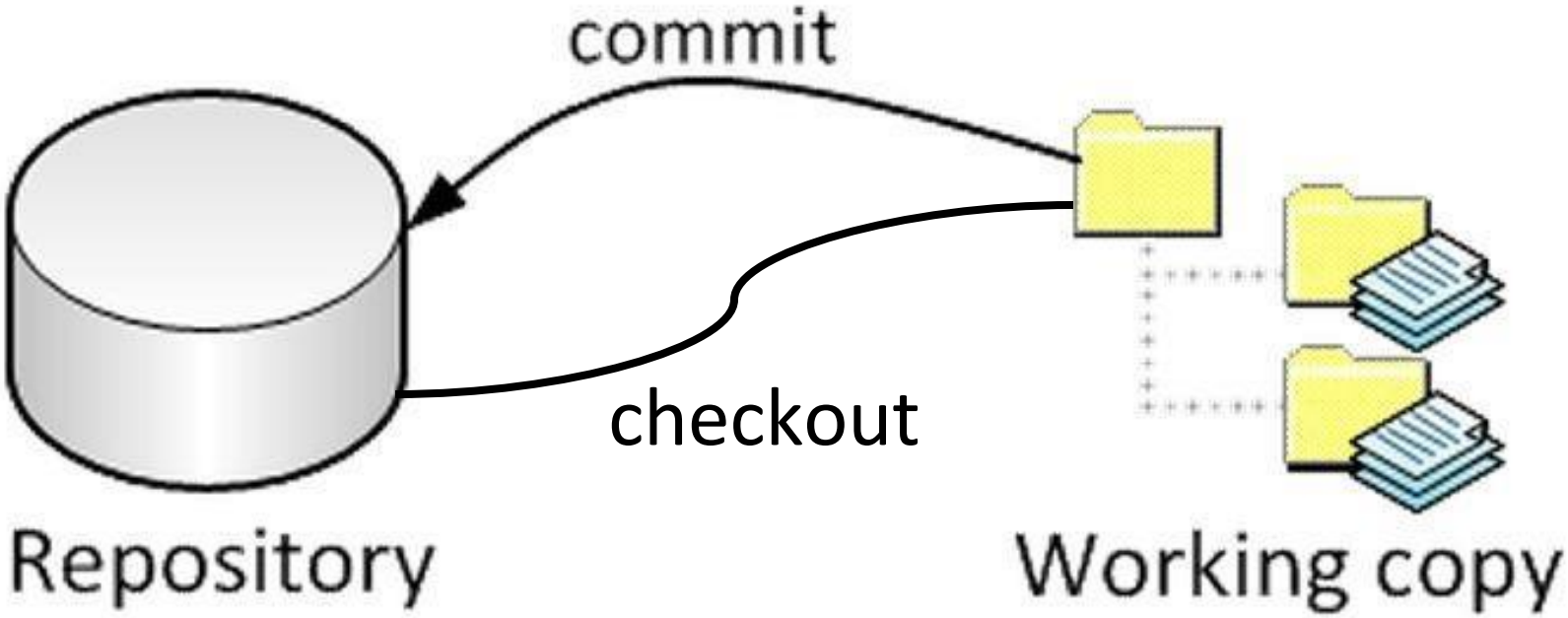
Cite: https://git-scm.com/book/en/v2/Getting-Started-Git-Basics

Git source control

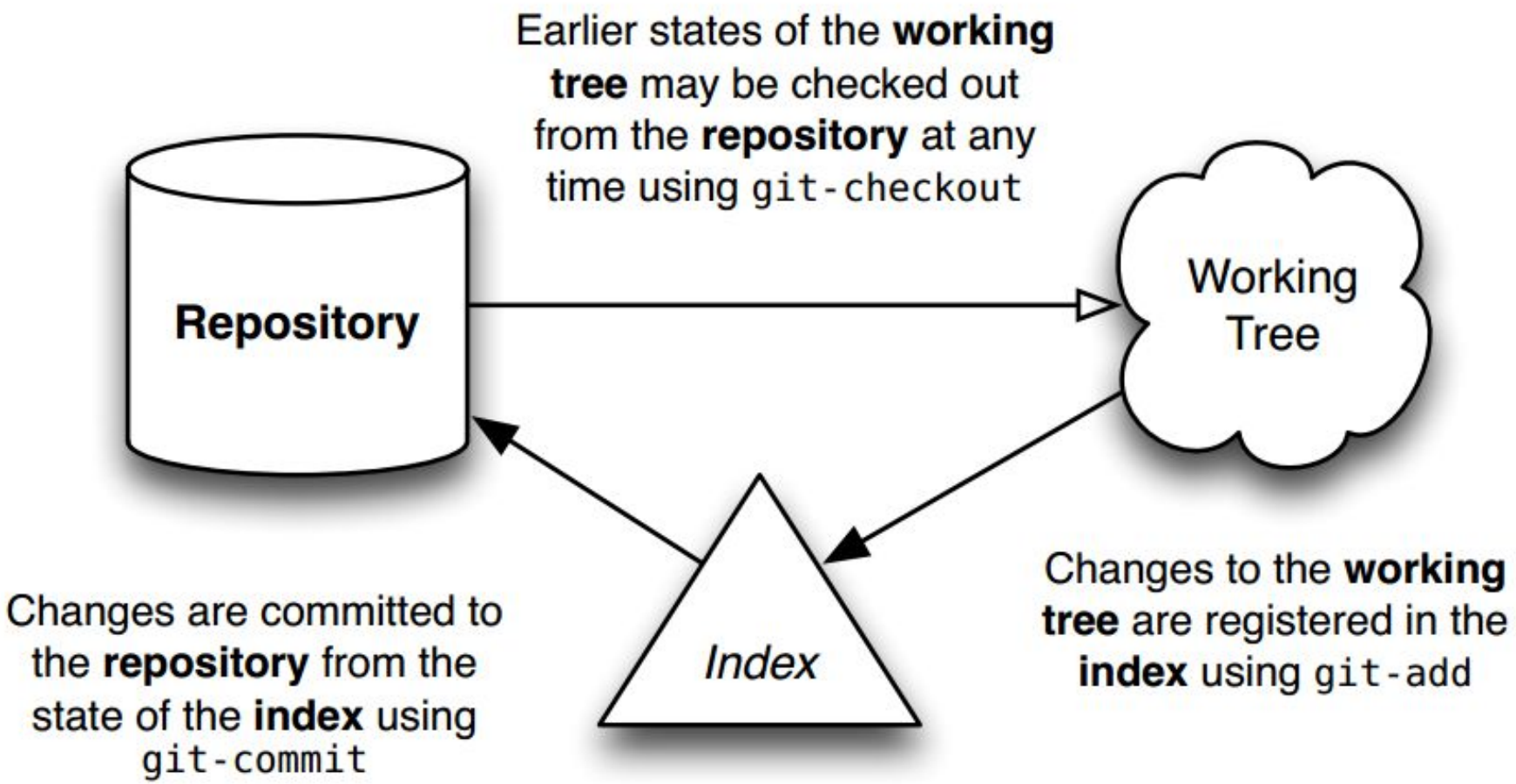
Terminology

- **Repository**
  - Files and folders related to the software code
  - Full history of the software
- **Working copy**
  - Copy of software's files in the repository
- **Check-out**
  - To create a working copy of the repository
- **Check-in/Commit**
  - Write the changes made in the working copy to the repository
  - Commits are recorded by the SCS

Big Picture in local machine



Git Local Workflow



Git States

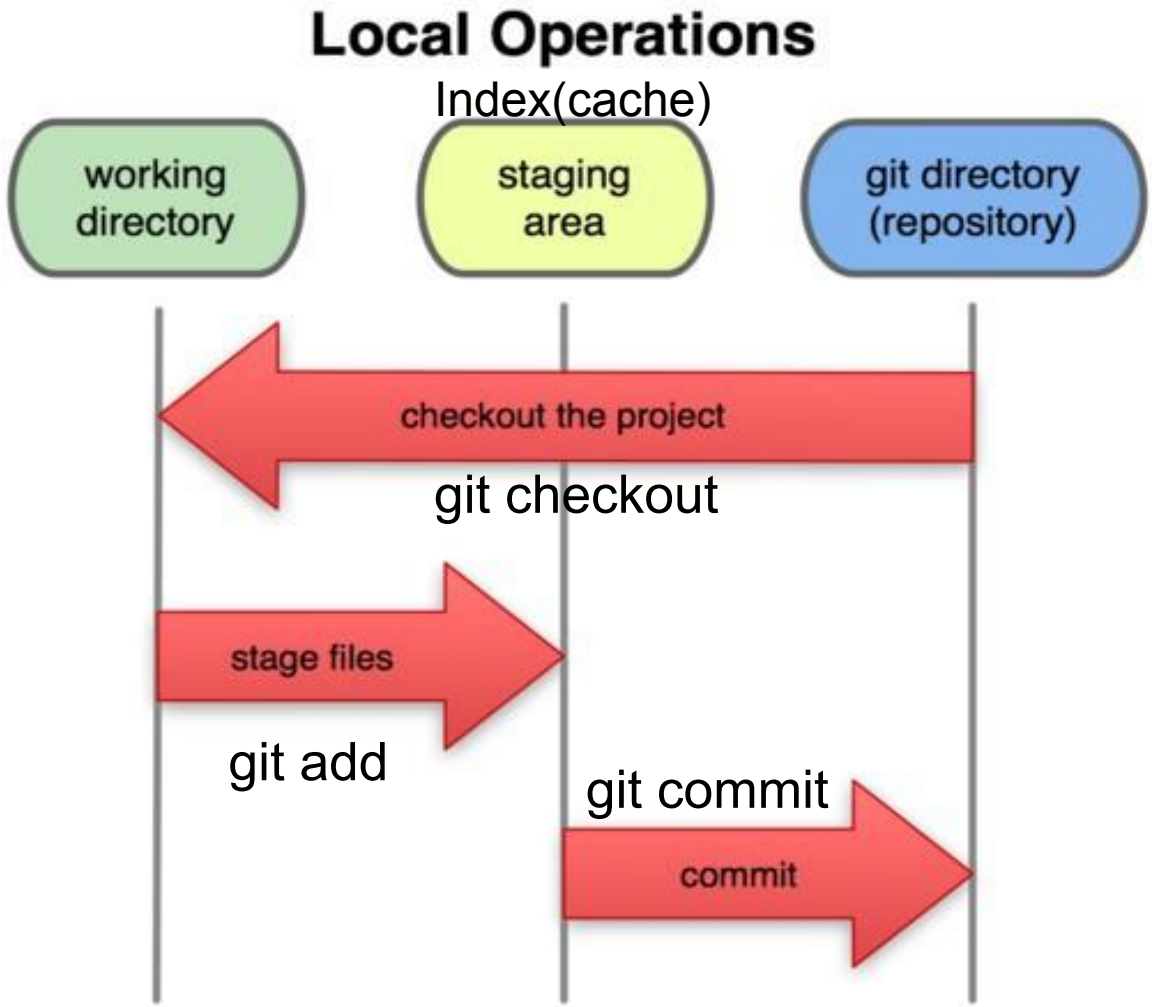


Image Source: git-scm.com

First Git Repository

- `$ mkdir gitroot`
- `$ cd gitroot`
- `$ git init`
  - creates an empty git repo (.git directory with all necessary subdirectories)
- `$ echo "Hello World" > hello.txt`
- `$ git add .`
  - Adds content to the index
  - Must be run prior to a commit
- `$ git commit –m 'Check in number one'`

Git commands

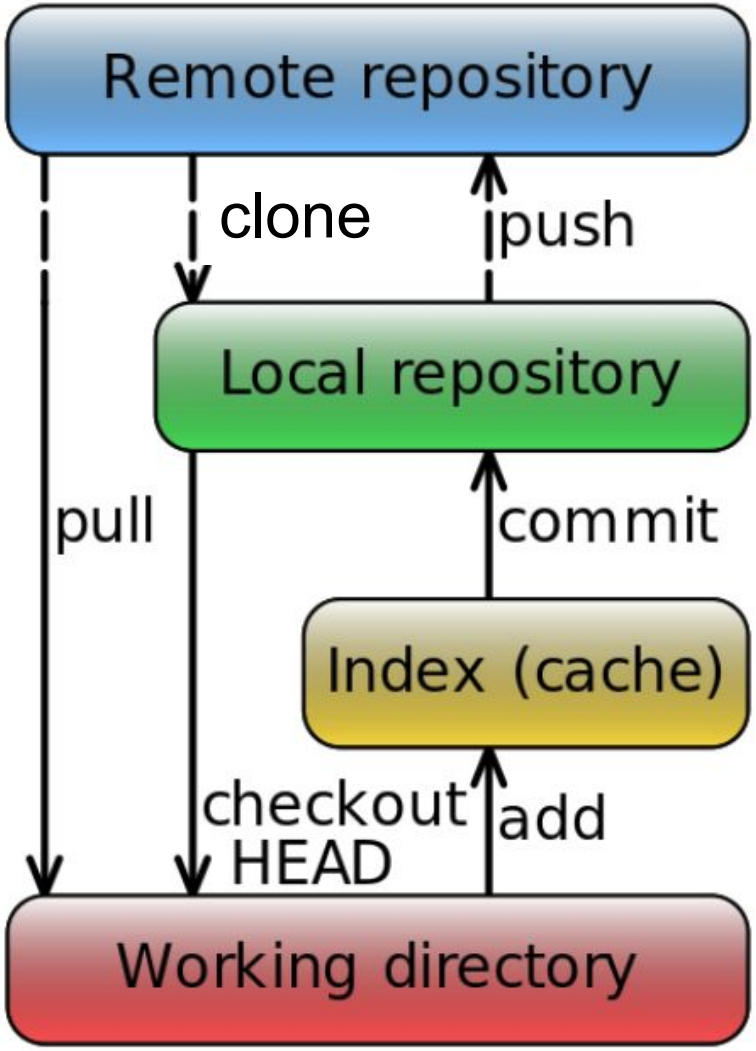
- Repository creation
  - **git init** (start a new repository)
  - **git clone** (create a copy of an existing repository)
- Branching
  - **git checkout <tag/commit> -b <new_branch_name>**

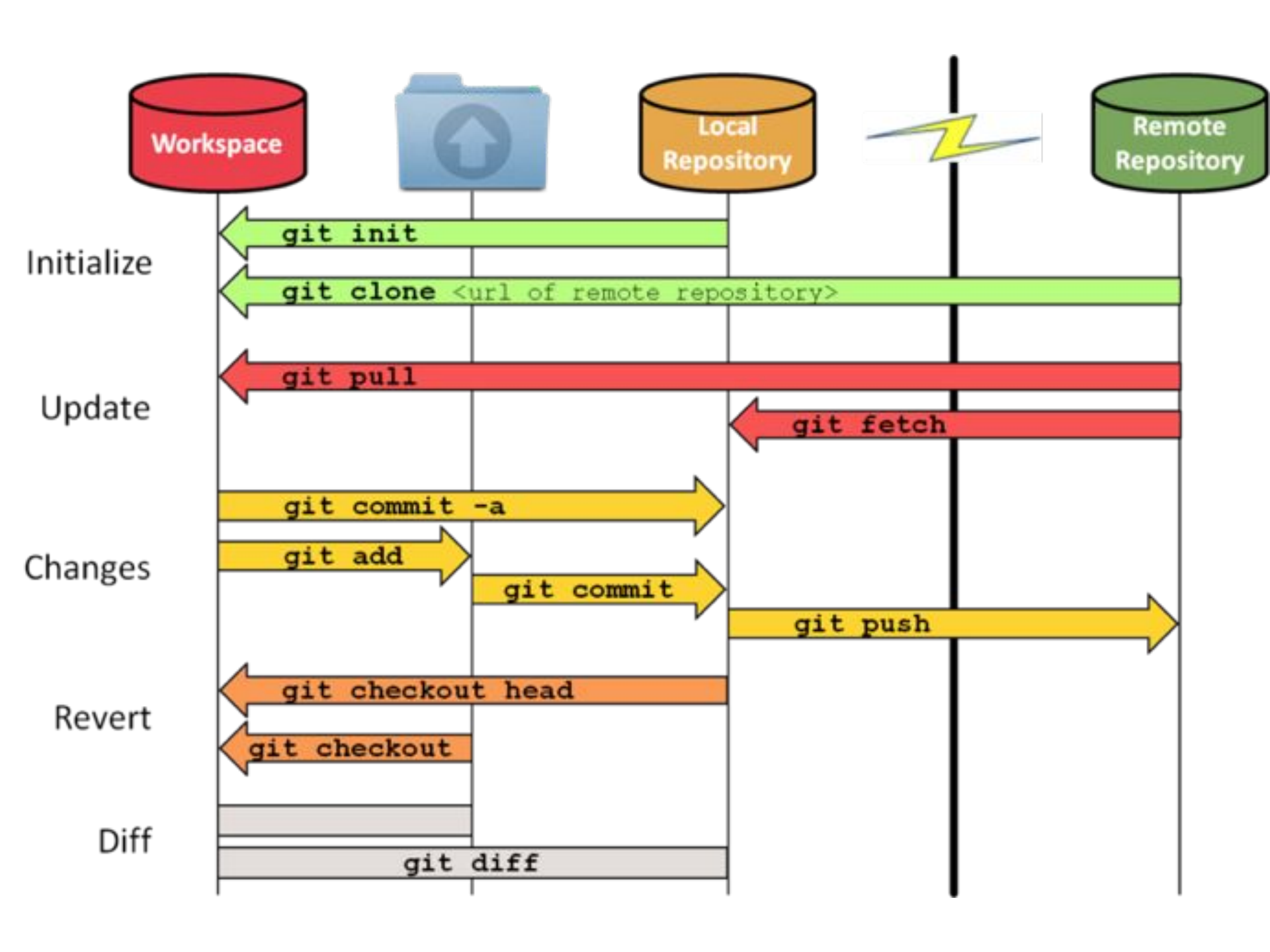
    (creates a new branch)
- Commits
  - **git add** (stage modified files)
  - **git commit** (check-in changes to the repository)
- Getting info
  - **git status** (shows modified files, new files, etc)
  - **git diff** (compares working copy with staged files)
  - **git log** (shows history of commits)
  - **git show** (show a certain object in the repository)
- Getting help
  - **git help**

Working With Git

- `$ echo "I love Git" >> hello.txt`
- `$ git status`
  - Shows list of modified files
  - hello.txt
- `$ git diff`
  - Shows changes we made compared to index
- `$ git add hello.txt`
- `$ git diff`
  - No changes shown as diff compares to the index
- `$ git diff HEAD`
  - Now we can see changes in working version
- `$git commit –m 'Second commit'`

Overview from whole system




Workspace
Local Repository
Remote Repository
Initialize
git init
git clone <url of remote repository>
Update
git pull
git fetch
Changes
git commit -a
git add
git commit
git push
Revert
git checkout head
git checkout
Diff
git diff

Second Git repository

- Git clone <repo>
- initializes a .git directory inside it
- pulls down all the data for that repository
- checks out a working copy of the latest version

- Try the example in the Lab

Lab 4

- GNU Diffutils uses " ` " in diagnostics
  - Example: diff . –
  - Output: diff: cannot compare - to a directory
  - Want to use apostrophes only
- Diffutils maintainers have a patch for this problem

  maint: quote 'like this' or "like this", not `like this'
- Problem: You are using Diffutils version 3.0, and the patch is for a newer version

Lab 4

- Task: Fix an issue with the diff diagnostic
- Crucial Steps: first create a new work directory
  - (4) Generate a patch
    - First get the hash in the log file
    - Then use git format-patch -1 [hash code] \ --stdout > [the patch file]
  - (9) learn the detailed usage of vc-diff and vc-revert
  - (10) consider changing ` to ‘

Useful Links

- Git tutorial
- Git Beginner's Tutorial with testing terminal
- Git Cheat Sheet
- Git from the bottom up
- Putty X11 forwarding

  (you'll need this for gitk)
- Use gitk to understand git