Version (Source Code) Control

SWEN-250

Overview

Motivation – why is version control useful?

Key concepts

Variations on the basic theme

Example version control systems

Motivation

Progress, far from consisting in change, depends on retentiveness. Those cannot remember the past are condemned to repeat it.

- George Santayana

Motivation Scenario: I *think* this will work

Often we want to try out a change
- Trying a new algorithm or data structure
- Reorganizing code for clarity
- Experimenting with a half-cocked idea
- Seeing if the language works like you expect

It's a lot easier if you can perform such experiments *confidently*.
- That is, you can get back to where you started
- VCS can provide a virtual trail of breadcrumbs
- If you botch things, you can return to a stable state

Motivation Scenario: How did I get here?

Like waypoints on a GPS system.

- Allows you to track progress
- You can see how your program evolved and grew, step-by-step
- You can see where you made mistakes and how long it took you to find and fix them

The fancy software engineering term is *traceability*.

- Important for scheduling, tracking, and planning
- Allows you to go back to a previous version (“hey, what did we ship for version 1.5.2.9.5?”)

Motivation Scenario: Reconstruct the Past

Teacher: “So, show me – what was the code like before you made this change?”

You: “Ummm...”

Motivation: Pragmatic Programmers

Tip 23

Always Use Source Code Control

So let it be written …

So let it be done!

Key Concepts*

A repository is a designated disk location (directory) where the files and "breadcrumbs" for a project are kept.

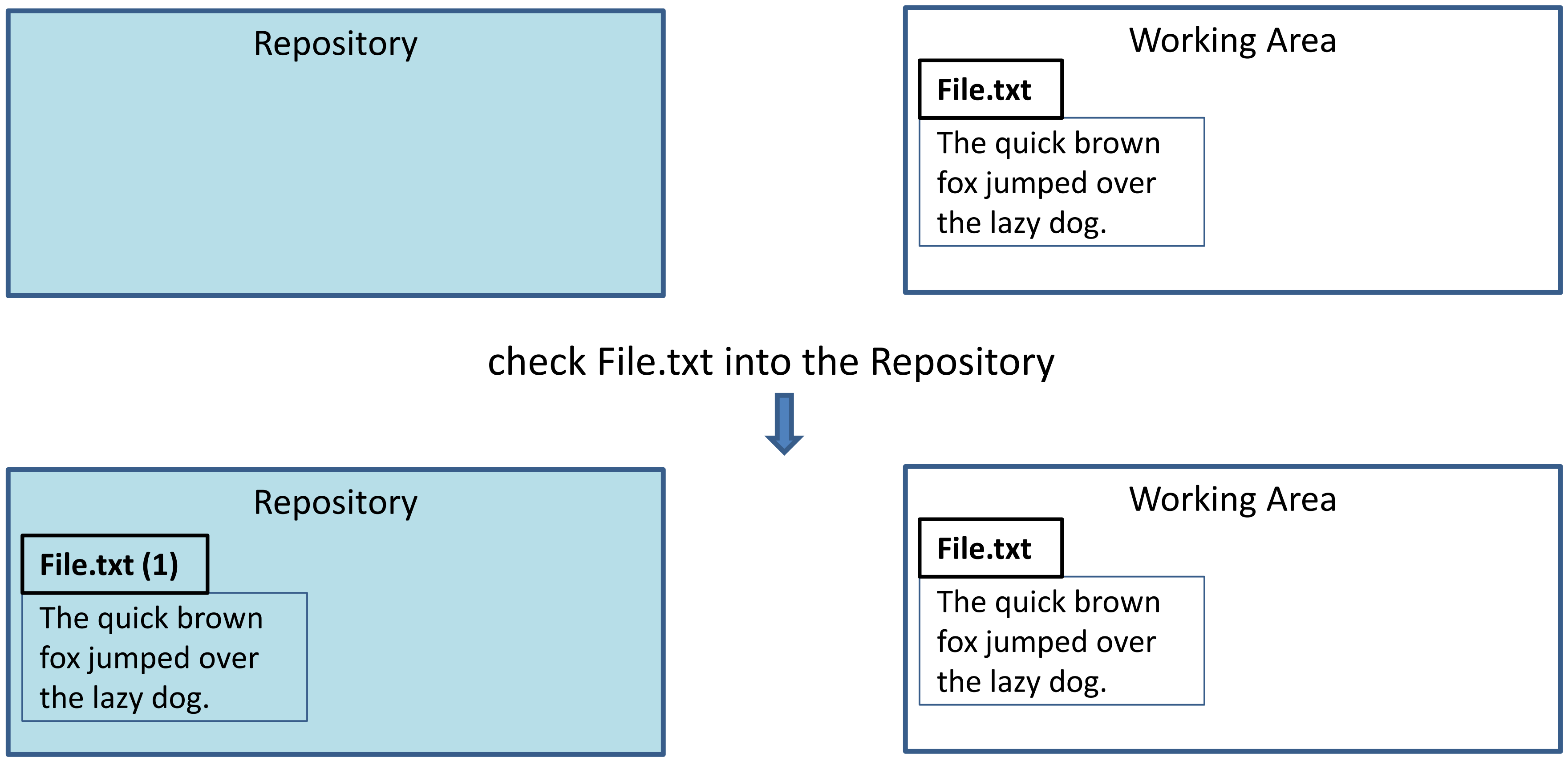
The work area is a location (directory) where the activities of editing, compiling, testing, etc. take place.

Files are periodically checked in to the repository from the work area, creating a new version.

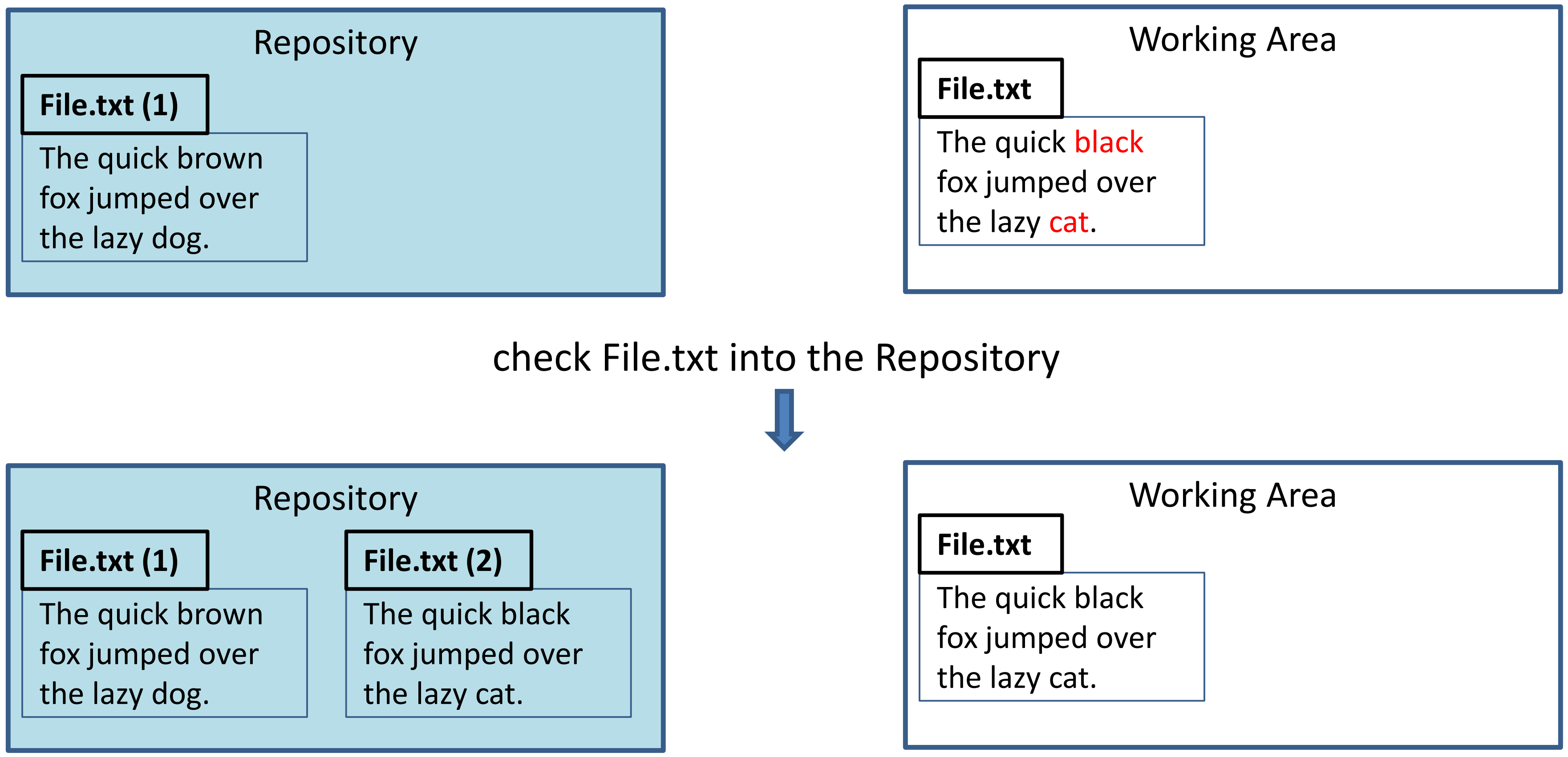
Files can be checked out of the repository (to start work on existing project, or to restore the state of the work area to a previous state).

* Terminology slightly different with git

Check a File into a Repository


Repository
Working Area
File.txt
The quick brown fox jumped over the lazy dog.
check File.txt into the Repository
Repository
File.txt (1)
The quick brown fox jumped over the lazy dog.
Working Area
File.txt
The quick brown fox jumped over the lazy dog.

Check a Changed File into a Repository



Repository

File.txt (1)

The quick brown
fox jumped over
the lazy dog.

Working Area

File.txt

The quick black
fox jumped over
the lazy cat.

check File.txt into the Repository

Repository

File.txt (1)

The quick brown
fox jumped over
the lazy dog.

File.txt (2)

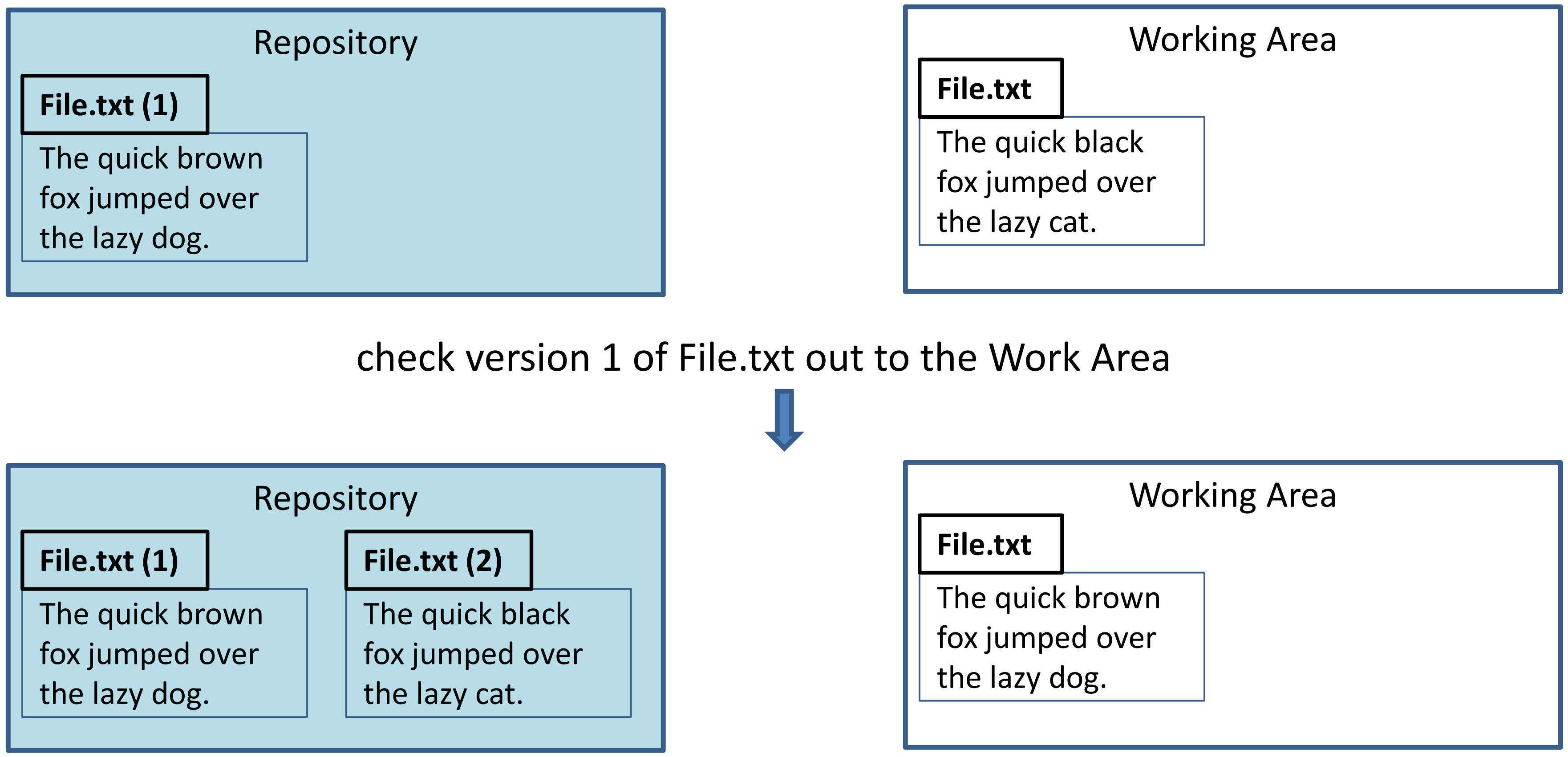
The quick black
fox jumped over
the lazy cat.

Working Area

File.txt

The quick black
fox jumped over
the lazy cat.

Revert to a Previous Version in a Repository


Repository
File.txt (1)
The quick brown fox jumped over the lazy dog.
Working Area
File.txt
The quick black fox jumped over the lazy cat.
check version 1 of File.txt out to the Work Area
Repository
File.txt (1)
The quick brown fox jumped over the lazy dog.
File.txt (2)
The quick black fox jumped over the lazy cat.
Working Area
File.txt
The quick brown fox jumped over the lazy dog.

A Potential Problem

We’ll have multiple copies of File.txt that are almost the same –won’t this waste space?

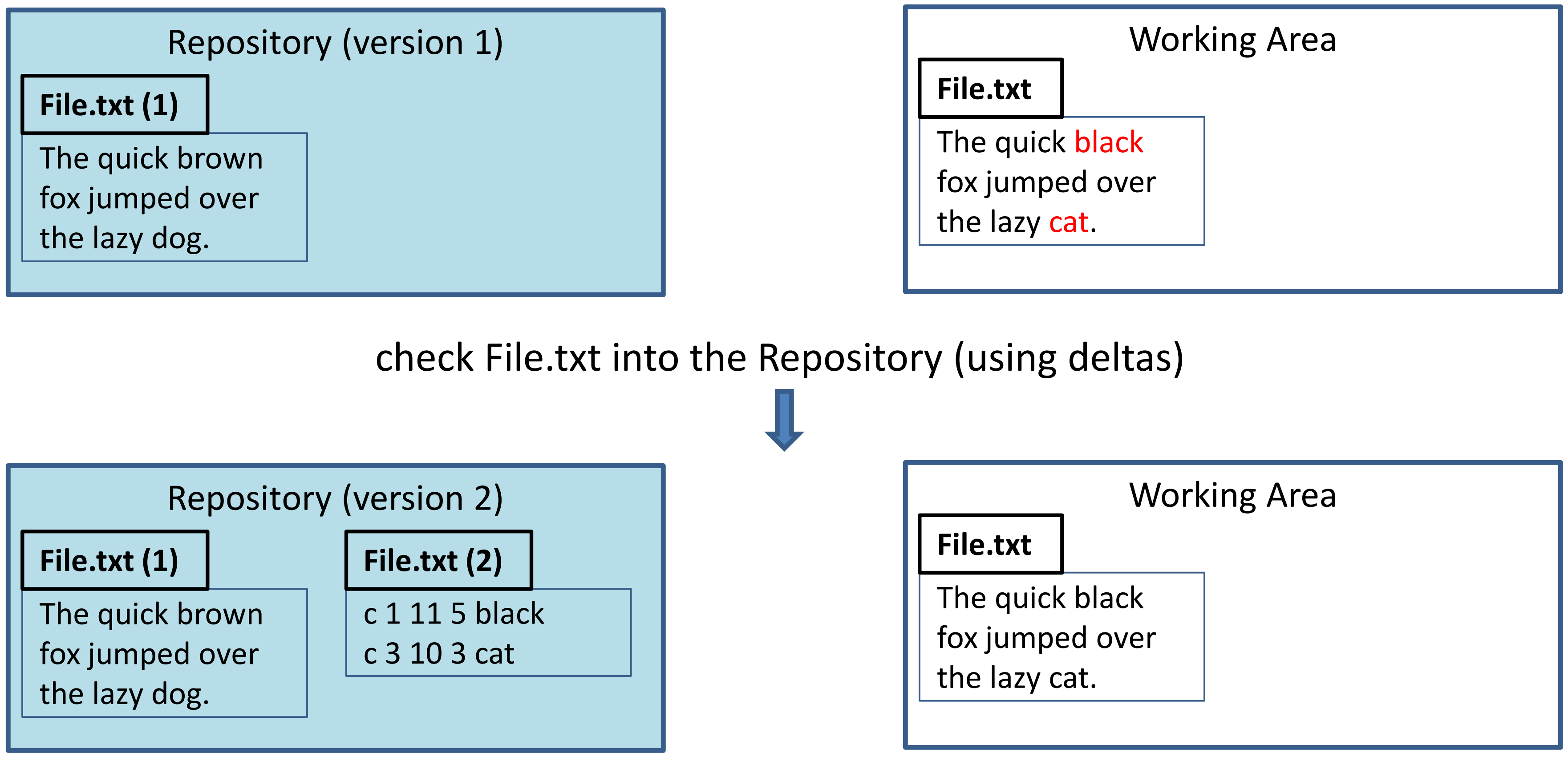
Yes – unless we use *deltas*.

- Usually changes from version to version are small.
- We can save space by only saving the changes (deltas).
- Basically, we need additions, deletions, changes.

Example: **a** *lno appended_text*
d *lno*
c *lno start length new_text*

With smart differences and compression, deltas become very small.

Check in a Changed File (w/deltas)


Repository (version 1)
File.txt (1)
The quick brown fox jumped over the lazy dog.
Working Area
File.txt
The quick black fox jumped over the lazy cat.
check File.txt into the Repository (using deltas)
Repository (version 2)
File.txt (1)
The quick brown fox jumped over the lazy dog.
File.txt (2)
c 1 11 5 black
c 3 10 3 cat
Working Area
File.txt
The quick black fox jumped over the lazy cat.

File vs. Repository Versioning

Versioning by file:

- Each file in repository has its own version number.
- Frequently changed files have higher numbers than stable files.
- May be difficult to find all the individual files representing one logical version.

Versioning by repository:

- Any changes update the version number of the entire repository.
- Easy to find all files comprising a given system version.
- Harder to find specific version of a given file.

Centralized vs. Distributed Repositories

Centralized:

- One master directory.
- All changes (by any team member) are applied to the master.
- Difficult for individuals to leave bread crumbs for their own experiments.

Distributed:

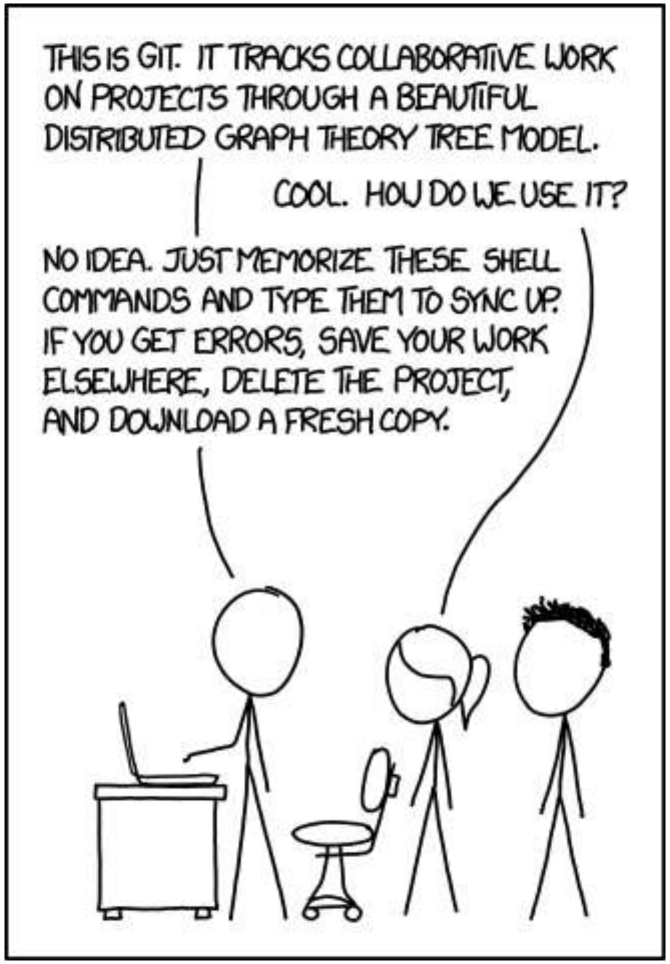
- Every developer has own repository.
- Changes are done to local repository.
- If working on a team, periodically PUSH local changes to designated central repository.

Sample of Version Control Systems

- CVS – Concurrent Versioning System
  - Centralized
  - File versioning
  - Used in CS3
- git – from Linus Torvalds, creator of Linux
  - Decentralized
  - Repository versioning
  - Used in this course
- Some others you may encounter
  - SVN – Subversion: Centralized, repository versioned
  - TFS – Team Foundation Services: Centralized & Distributed, file and repo versions
  - RCS – Revision Control System: Centralized, file versioned
  - SCCS – Source Code Control System: Centralized, file versioned

Git

Understanding the machinery to whittle away the uncertainty


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.

Been here before? (web comic by XKCD)

Git vocabulary

repository: a place for storing things aka repo. With Git, this means your code folder
clone: Copy all files from a repo to your local drive
head: A "pointer" to the latest code you were working on
add: An action to ask Git to track a file
commit: An action to save the current state to prepare for next step (e.g. push to repo)
remote: A repository that isn't local. Can be in another folder or in the cloud (for example: Github or gitlab): helps other people to easily collaborate, as they don't have to get a copy from your system — they can just get it from the cloud. Also, ensures you have a backup in case you break your laptop
pull: An action to get updated code from the remote
push: An action to send updated code to the remote
merge: An action to combine two different versions of code
status: Displays information about current repository status
log: Show the history of the repo actions

Where is everything?

```
$ tree .git/
.git/
├── HEAD
├── config
├── description
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   ├── pre-receive.sample
│   ├── prepare-commit-msg.sample
│   └── update.sample
├── info
│   └── exclude
├── objects
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags
8 directories, 14 files
```

Introducing the magic controlled by a hidden folder: .git/
In every git repository, you'll see something like this

A good reference

- https://medium.freecodecamp.org/how-not-to-be-afraid-of-git-anymore-fe1da7415286