A Simple Introduction to Git: a distributed version-control system

CS 5010 Program Design Paradigms "Bootcamp"

Lesson 0.5

Learning Objectives

- At the end of this lesson you should be able to explain:
  - how git creates a mini-filesystem in your directory
  - what pull, commit, and push do
  - the elements of the basic git workflow
  - how git allows you to work across multiple computers

Git is a distributed version-control system

- You keep your files in a *repository* on your local machine.
- You synchronize your repository with a repository on a server.
- If you move from one machine to another, you can pick up the changes by synchronizing with the server.
- If someone else on your team uploads some changes to your files, you can pick those up by synchronizing with the server.

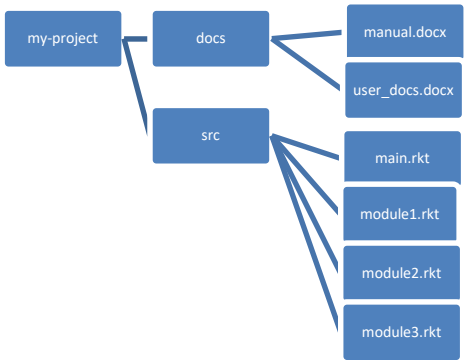
Git is a distributed version-control system

- Terminology: In git-speak, a “version” is called a “commit.”
- Git keeps track of the history of your commits, so you can go back and look at earlier versions, or just give up on the current version and go back some earlier version.

A simple model of git

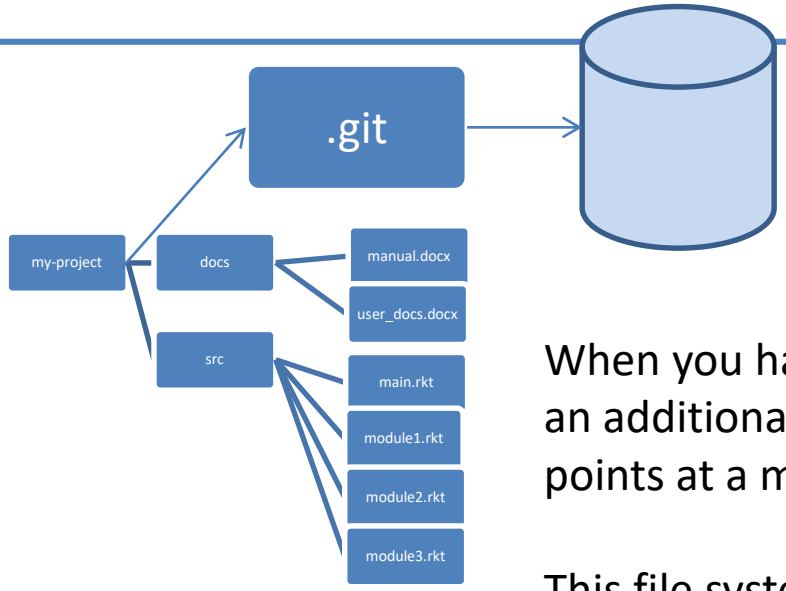
- Most git documentation gets into details very quickly.
- Here's a very simple model of what's going on in git.

Your files



Here are your files, sitting in a directory called my-project

Your files in your git repository

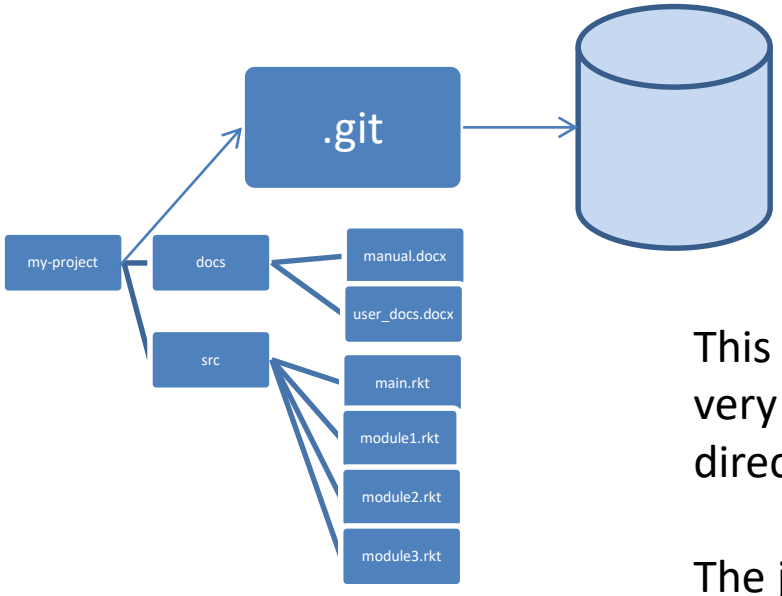


When you have a git repository, you have an additional directory called .git, which points at a mini-filesystem.

This file system keeps all your data, plus the bells and whistles that git needs to do its job.

All this sits on your local machine.

The git client



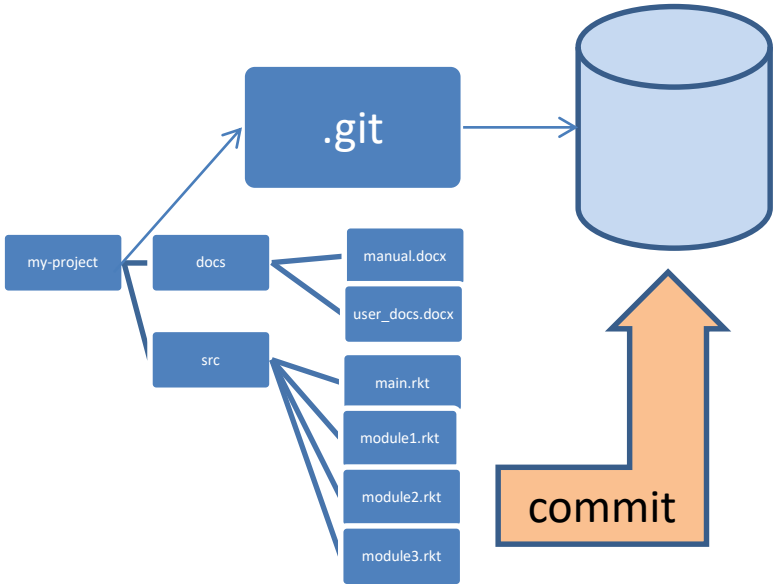
This mini-filesystem is highly optimized and very complicated. Don't try to read it directly.

The job of the git client is to manage this for you.

Your workflow (part 1)

- You edit your local files directly.
  - You can edit, add files, delete files, etc., using whatever tools you like.
  - This doesn't change the mini-filesystem, so now your mini-filesystem is behind.

A Commit



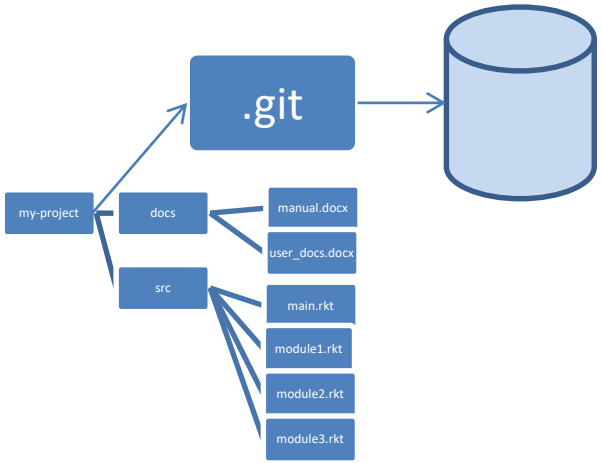
When you do a “commit”, you record all your local changes into the mini-filesystem.

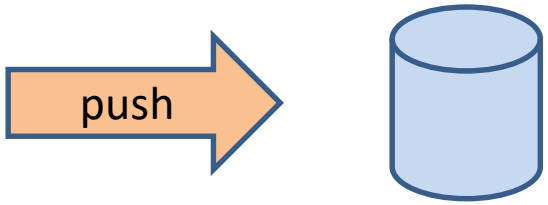
The mini-filesystem is “append-only”. Nothing is ever over-written there, so everything you ever commit can be recovered.

Synchronizing with the server (1)

your local machine

a server, somewhere on the internet, eg. github.com



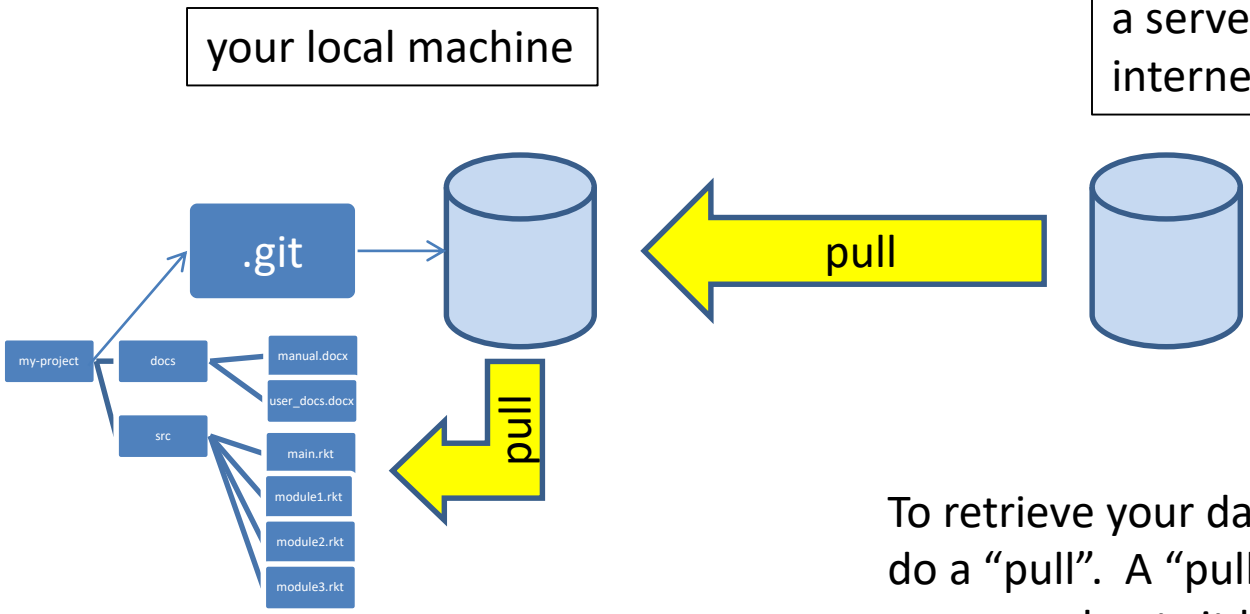


At the end of each work session, you need to save your changes on the server. This is called a "push".

Now all your data is backed up.

- You can retrieve it, on your machine or some other machine.
- We can retrieve it (that's how we collect homework)

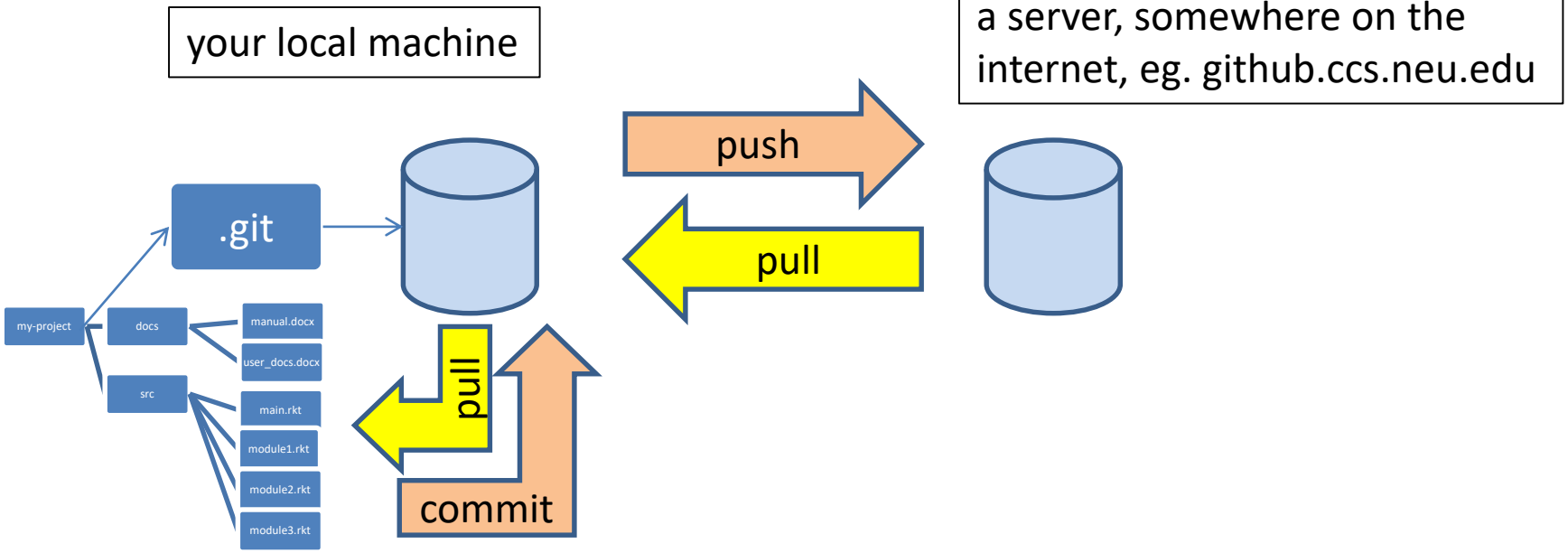
Synchronizing with the server (2)



To retrieve your data from the server, you do a “pull”. A “pull” takes the data from the server and puts it both in your local mini-fs and in your ordinary files.

If your local file has changed, git will merge the changes if possible. If it can’t figure out how to the merge, you will get an error message. We’ll talk about this some more a little later in this lesson.

The whole picture

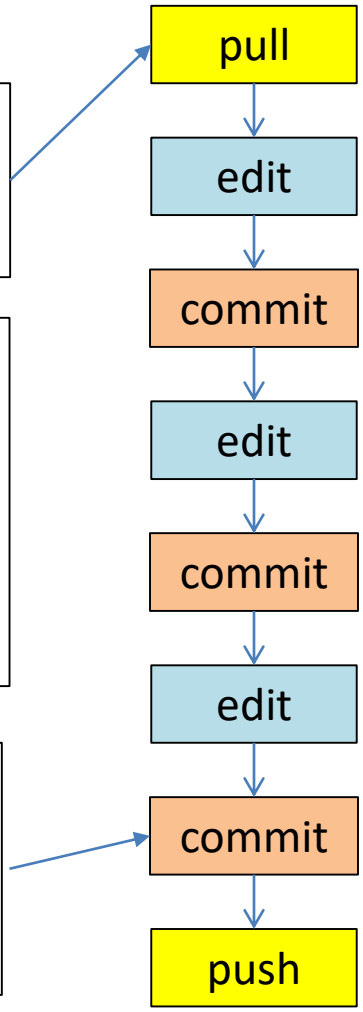


Your workflow for a session

Start by pulling the latest version of your work from the server

Best practice: commit your work whenever you've gotten one part of your problem working, or before trying something that might fail.

At the end of each session:
(1) Update *your-ID*-log.txt
(2) Commit
(3) push



If you want to abandon your changes, you can always go back to your preceding commit, using "git checkout". (Go read about "git checkout", which does lots of useful things…)

Submitting a Work Session log

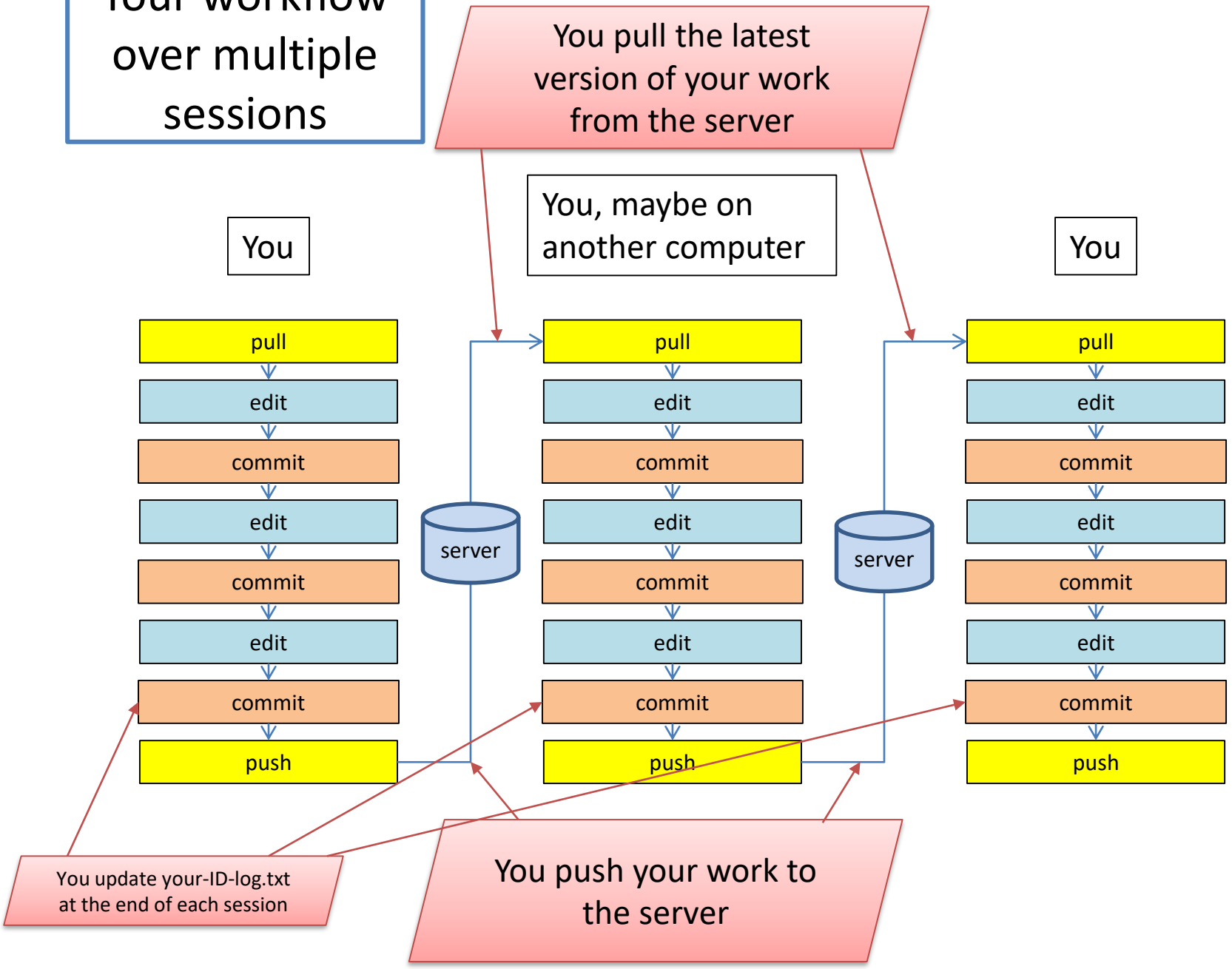
- At the end of your work session, update a file called *your-ID*-log.txt (replace "your-ID" with your CCIS login ☺)
- This file records the time you spent in this work session.

```
0 hours 0 minutes spent this session
2017-01-06
I hereby certify that all the work in this commit is my own except for
materials distributed as part of this class.
/* do not change the format of anything above this line */

After each work session, please update and commit this file (with
'yourID' replaced by your CCIS login name), updating the first two
lines with the amount of time you spent working and the date when you
stopped.  You may also record any notes you wish to make about what
you did during the session.

Every commit of this file constitutes a work log entry.  Each entry
should replace the previous one, so there will only be one session
recorded at any particular version of this file.  Do *not* change the
format of the first four lines, as the course staff will use
automated tools to track the amount of time students spend on each
problem set.
```

Your workflow over multiple sessions



Oh no! I've got a git conflict

- If you update your work before you pull, git will try to merge the changes when you do pull.
- If git can't figure out how to do the merge, it will go into a special mode in which it expects you to resolve the conflicts.
- When you have the files the way you want them, you can commit them and continue working. Here's a guide.
- Since you will not be working in pairs, this is unlikely to happen too often.

Other ways to use git and github

- There are lots of possible ways to use git and github.
- We only care about the “master” branch
- If you know git well, and you want to do something fancier with multiple branches, merges, and whatnot, feel free to do so.
- But you should be able to get by just fine with just a single master branch.

We believe in the KISS principle:
“Keep It Simple, Stupid!”

Summary

- In this lesson you have learned
  - that git creates a mini-filesystem in your directory
  - what pull, commit, and push do
  - the elements of the basic git workflow
  - how git allows you to work across multiple computers.

Next Steps

- There are many interactive git tutorials on the web.
  - Go find one and do it.
  - If you find one you like, recommend it to your classmates on the Discussion Board.
- If you have questions about this lesson, ask them on the Discussion Board
- Go on to the next lesson