Data Management for Reproducable Research

Thomas R. Cook

23-OCT 2015

Long-term reproducability and Mysterious Data:

- Common Scenario:
- Get novel data file
  - Make some changes to it
  - Save over original file

Long-term reproducability and Mysterious Data:

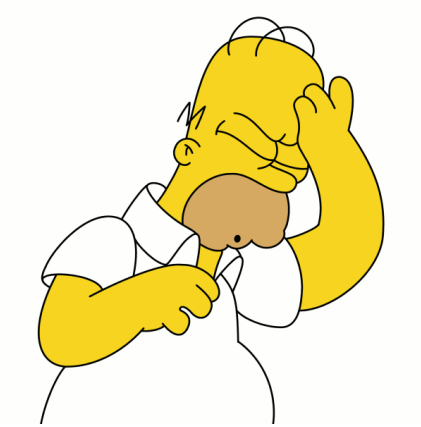
- Common Scenario:
- Get novel data file
  - Make some changes to it
  - Save over original file

Six months later...

- Return to project...
  - What does `log_inerv_1234.b` mean? How did I get it? Why is it driving my results?
- Even worse if someone asks for your replication data
  - You need to be able to explain how you arrived at a given variable/model/etc
- Moar worse if your co-author created `log_inerv_1234.b`

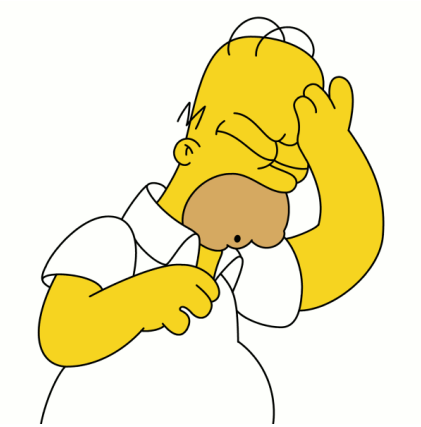
Six months later...

- Return to project...
  - What does `log_inerv_1234.b` mean? How did I get it? Why is it driving my results?
- Even worse if someone asks for your replication data
  - You need to be able to explain how you arrived at a given variable/model/etc
- Moar worse if your co-author created `log_inerv_1234.b`



Six months later...

- Return to project...
  - What does `log_inerv_1234.b` mean? How did I get it? Why is it driving my results?
- Even worse if someone asks for your replication data
  - You need to be able to explain how you arrived at a given variable/model/etc
- Moar worse if your co-author created `log_inerv_1234.b`



- Make R script, DO file or other script that generates data

Two big challenges with managing data

1. Track file changes over tiem
   - Long-term reproducibility
   - Version management
2. Collaboration with others

Common Solutions:

5. Edit data in-place (!)
6. Dropbox
7. Track Changes/time-machine
8. Email
9. New folder per version

That's a start...

But what about these other nightmare scenarios:

- Someone asks for old version of replication data

That's a start...

But what about these other nightmare scenarios:

- Someone asks for old version of replication data
- Coauthor deletes files in his/her dropbox folder

That's a start...

But what about these other nightmare scenarios:

- Someone asks for old version of replication data
- Coauthor deletes files in his/her dropbox folder
- You and co-author try to work on data at the same time (the dreaded `conflited copy`)

That's a start...

But what about these other nightmare scenarios:

- Someone asks for old version of replication data
- Coauthor deletes files in his/her dropbox folder
- You and co-author try to work on data at the same time (the dreaded `conflited copy`)
- Need to identify how project today differs from version 6 months ago

That's a start...

But what about these other nightmare scenarios:

- Someone asks for old version of replication data
- Coauthor deletes files in his/her dropbox folder
- You and co-author try to work on data at the same time (the dreaded `conflited copy`)
- Need to identify how project today differs from version 6 months ago
- Need to identify prior/abandoned approaches to analysis

That's a start...

But what about these other nightmare scenarios:

- Someone asks for old version of replication data
- Coauthor deletes files in his/her dropbox folder
- You and co-author try to work on data at the same time (the dreaded `conflited copy`)
- Need to identify how project today differs from version 6 months ago
- Need to identify prior/abandoned approaches to analysis
- The list goes on...

That's a start...

But what about these other nightmare scenarios:

- Someone asks for old version of replication data
- Coauthor deletes files in his/her dropbox folder
- You and co-author try to work on data at the same time (the dreaded `conflited copy`)
- Need to identify how project today differs from version 6 months ago
- Need to identify prior/abandoned approaches to analysis
- The list goes on...
- Git can help resolve all of these

Git is. . .

- Distributed Version Control System

Git is. . .

- Distributed Version Control System
  - Distributed $\rightarrow$ decentralized, many users

Git is. . .

- Distributed Version Control System
  - Distributed $\rightarrow$ decentralized, many users
  - Version Control $\rightarrow$ track changes, enable rollbacks,

Git is. . .

- Distributed Version Control System
  - Distributed $\rightarrow$ decentralized, many users
  - Version Control $\rightarrow$ track changes, enable rollbacks,
  - System $\rightarrow$ System

Git is. . .

- Distributed Version Control System
  - Distributed $\rightarrow$ decentralized, many users
  - Version Control $\rightarrow$ track changes, enable rollbacks,
  - System $\rightarrow$ System
- Think: "Track changes" on steroids

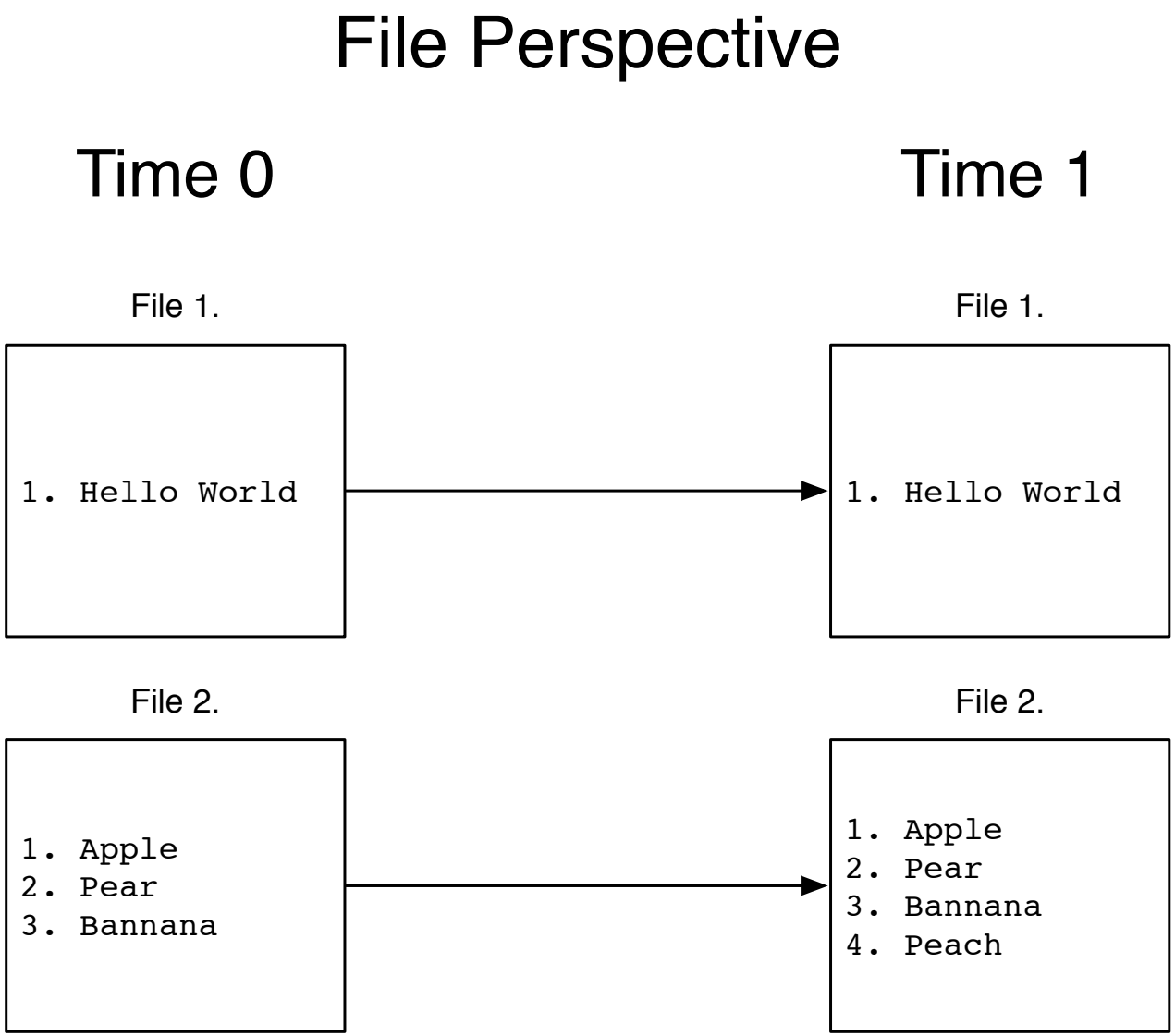
Repos

- Git tracks sets of files – multiple files at once
- Folder with set of files tracked by Git: Repository
  - Generally, a Git repo looks and works just like a folder
- Think: Repo project

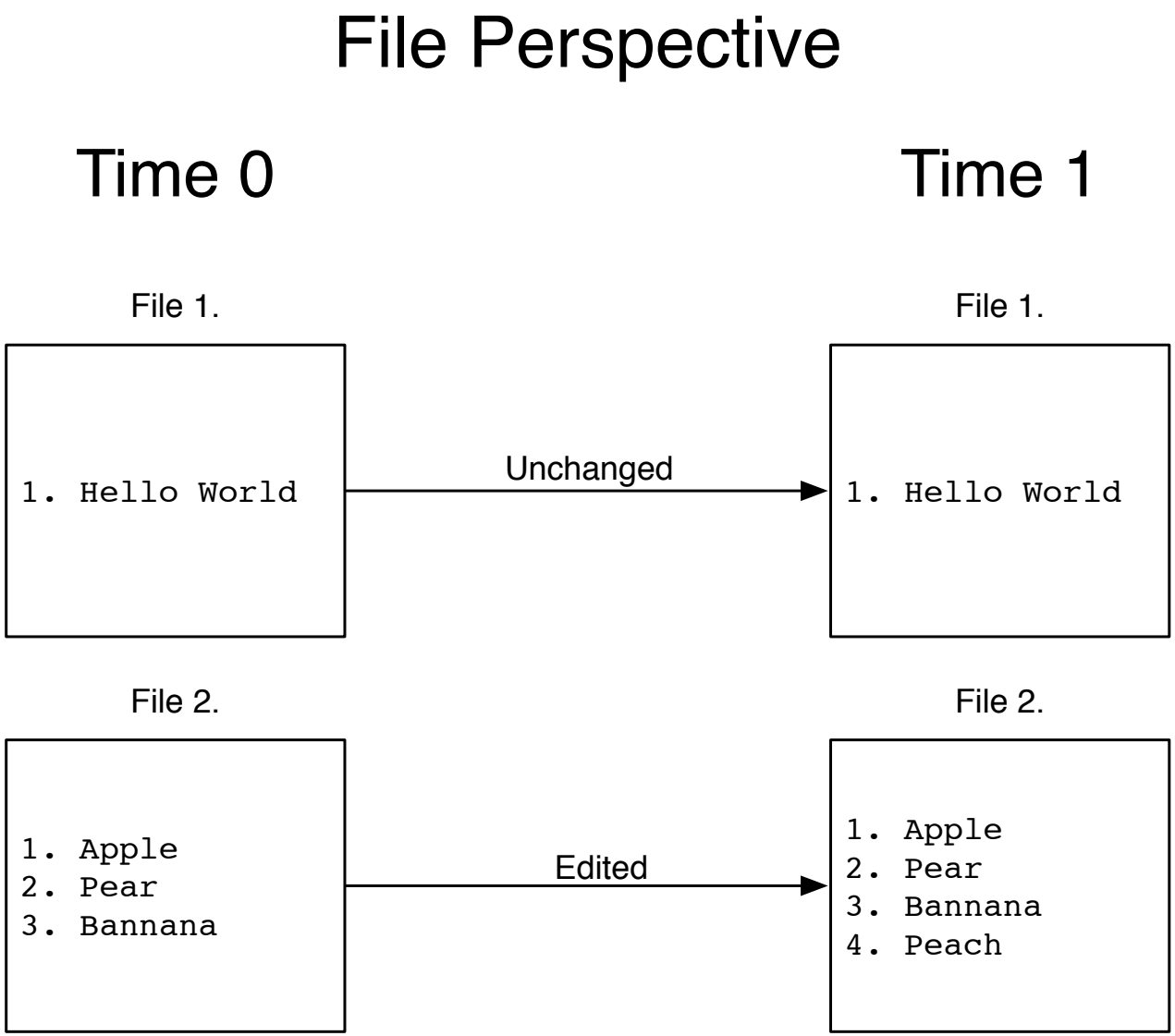
Commits

- A snapshot of (specified) files tracked by Git
  - Captures *changes* in specified files (since last commit)

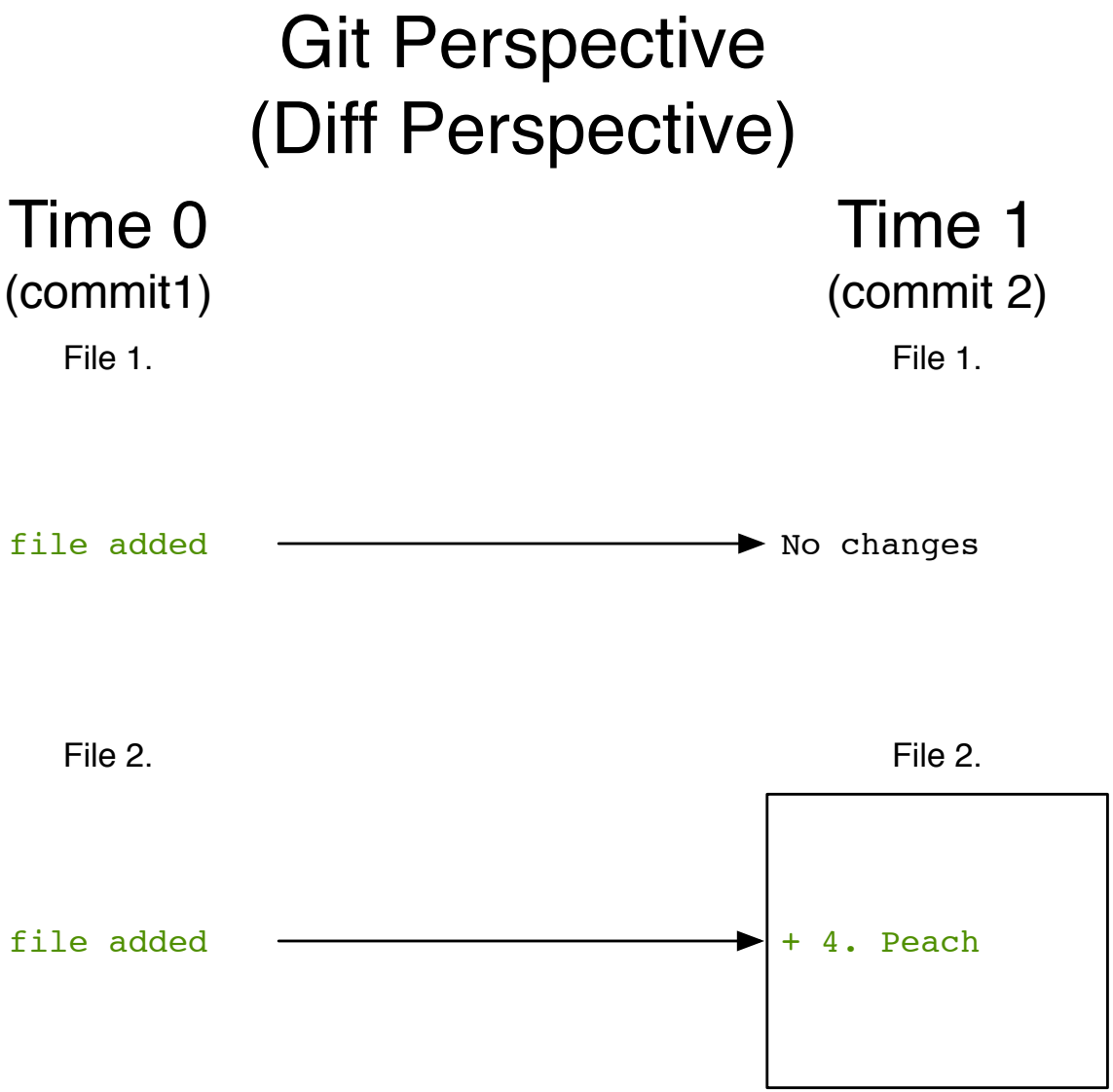
Commits


File Perspective
Time 0
Time 1
File 1.
1. Hello World
File 1.
1. Hello World
File 2.
1. Apple
2. Pear
3. Bannana
File 2.
1. Apple
2. Pear
3. Bannana
4. Peach

Commits


File Perspective
Time 0
Time 1
File 1.
1. Hello World
Unchanged
File 1.
1. Hello World
File 2.
1. Apple
2. Pear
3. Bannana
Edited
File 2.
1. Apple
2. Pear
3. Bannana
4. Peach

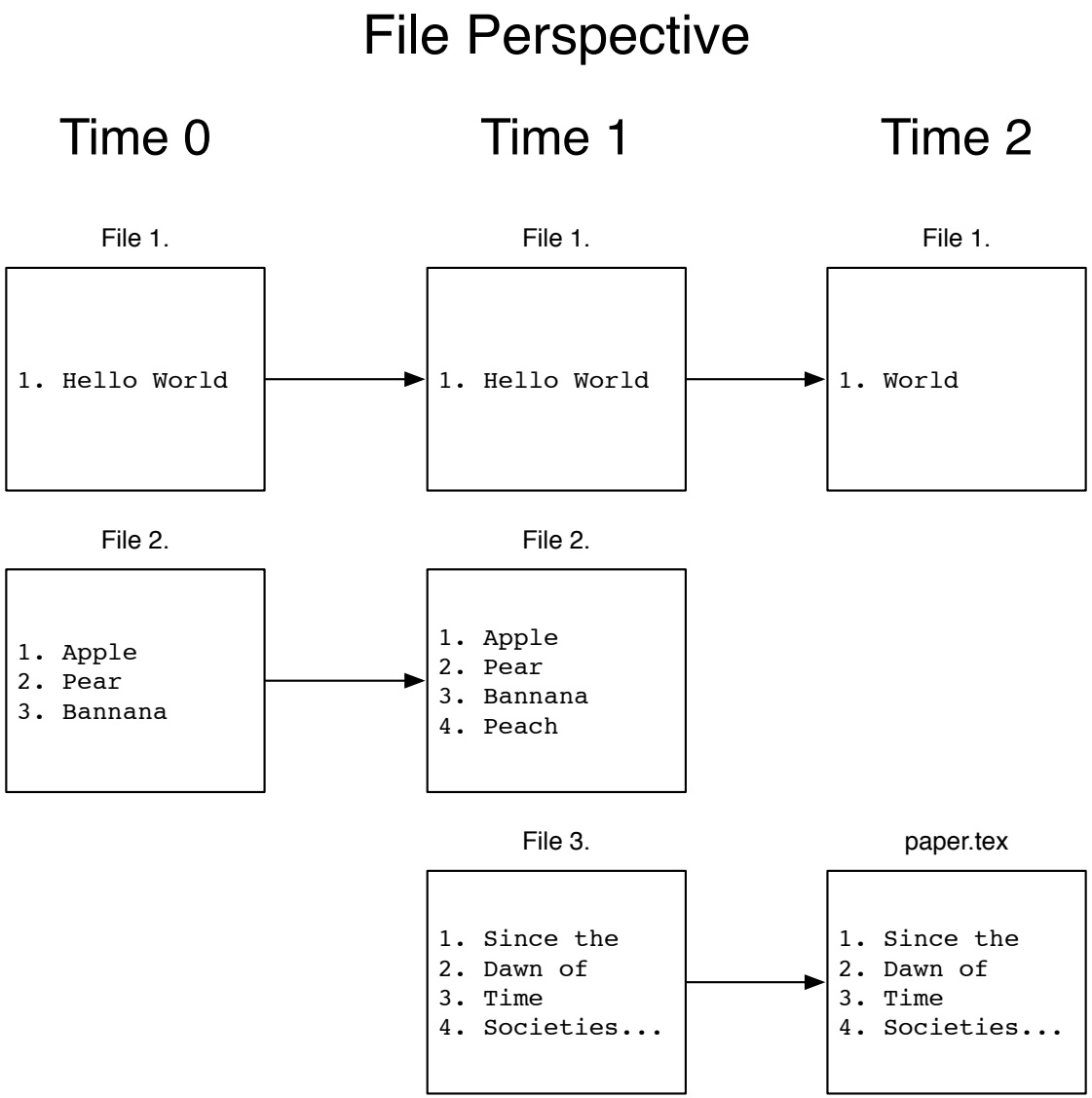
Commits


Git Perspective
(Diff Perspective)
Time 0
(commit1)
File 1.
Time 1
(commit 2)
File 1.
file added
No changes
File 2.
File 2.
file added
+ 4. Peach

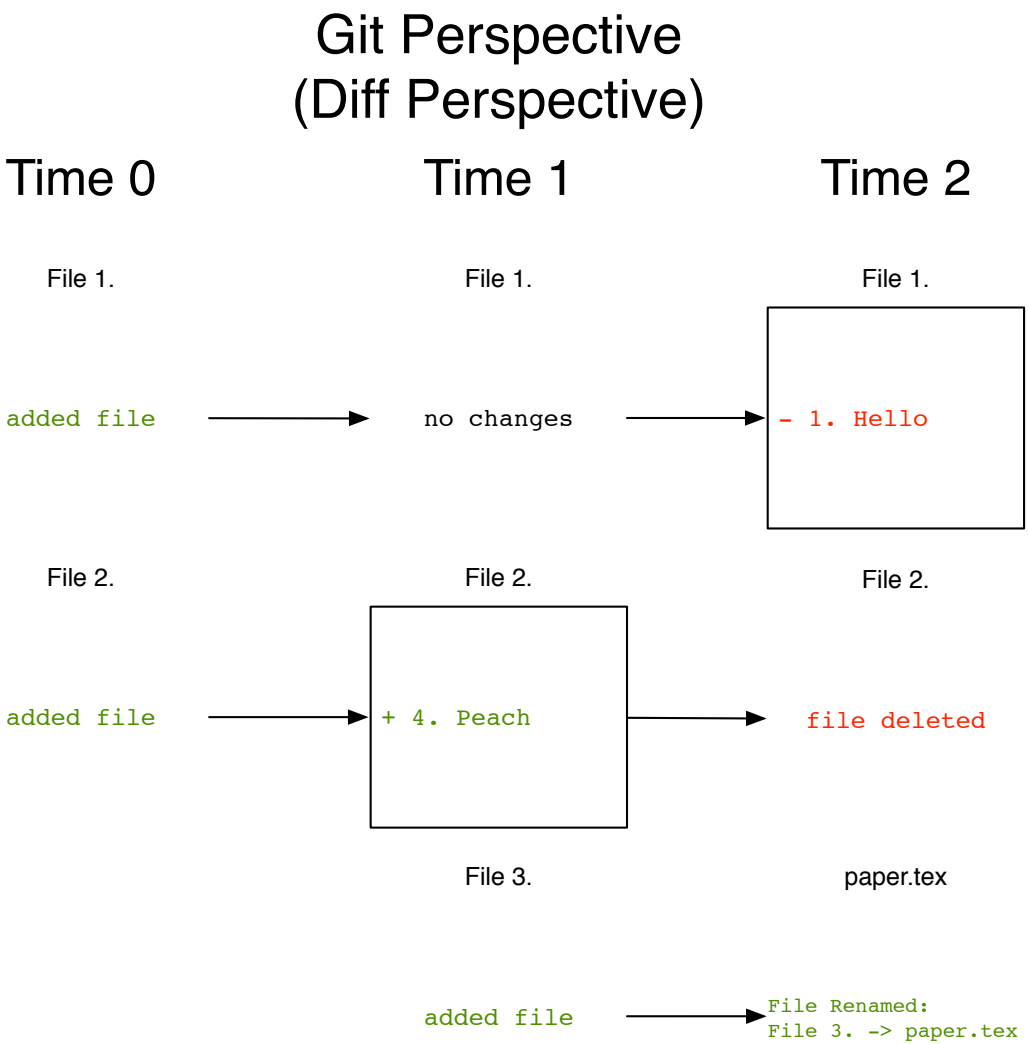
Commits

- A snapshot of (specified) files tracked by Git
  - Captures *changes* in specified files (since last commit)
  - Captures Files Added/Removed/Moved

Commits


File Perspective
Time 0
Time 1
Time 2
File 1.
1. Hello World
File 1.
1. Hello World
File 1.
1. World
File 2.
1. Apple
2. Pear
3. Bannana
File 2.
1. Apple
2. Pear
3. Bannana
4. Peach
File 3.
1. Since the
2. Dawn of
3. Time
4. Societies...
paper.tex
1. Since the
2. Dawn of
3. Time
4. Societies...

Commits


Git Perspective
(Diff Perspective)
Time 0
Time 1
Time 2
File 1.
File 1.
File 1.
added file
no changes
- 1. Hello
File 2.
File 2.
File 2.
added file
+ 4. Peach
file deleted
File 3.
paper.tex
added file
File Renamed:
File 3. -> paper.tex

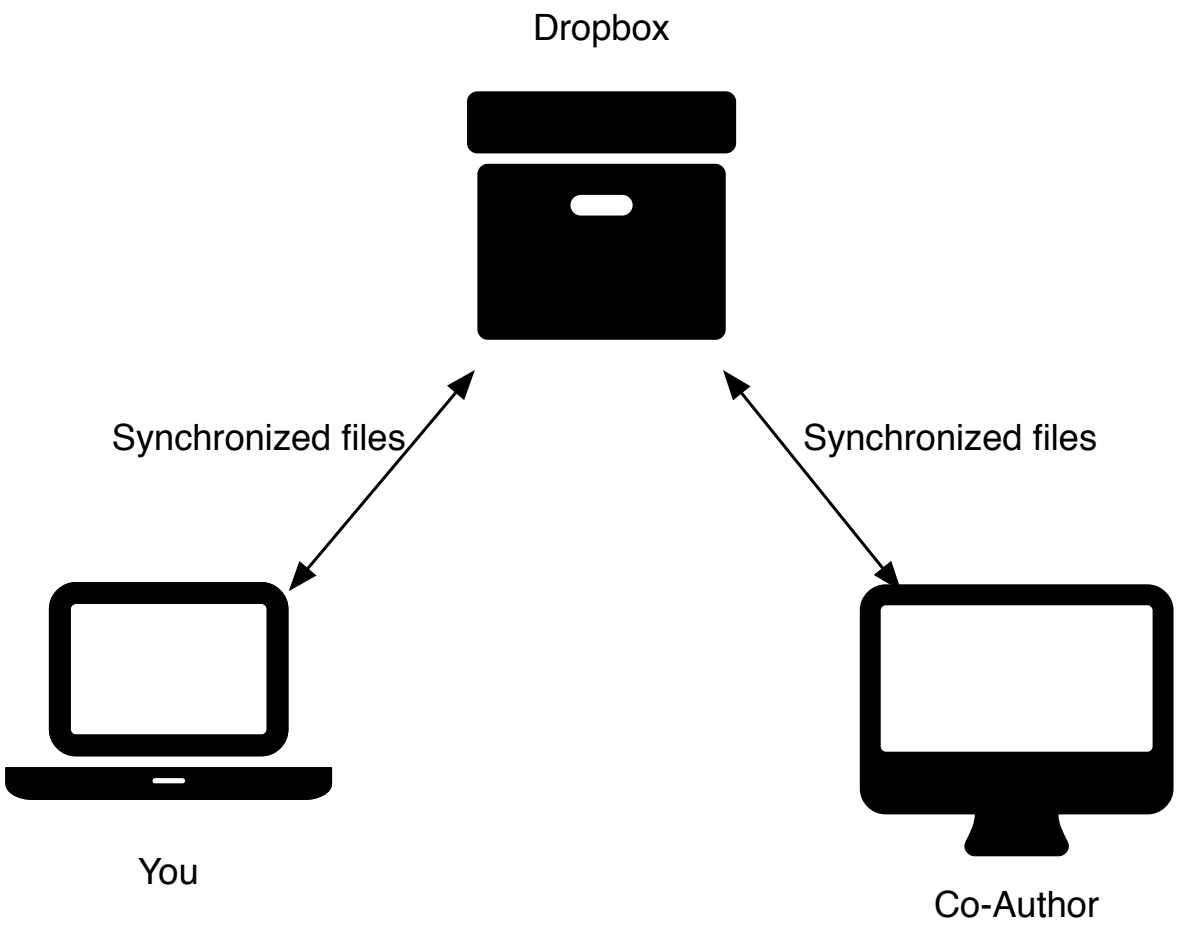
Commits

- A snapshot of (specified) files tracked by Git
    - Upshot: Can track file changes very closely over time

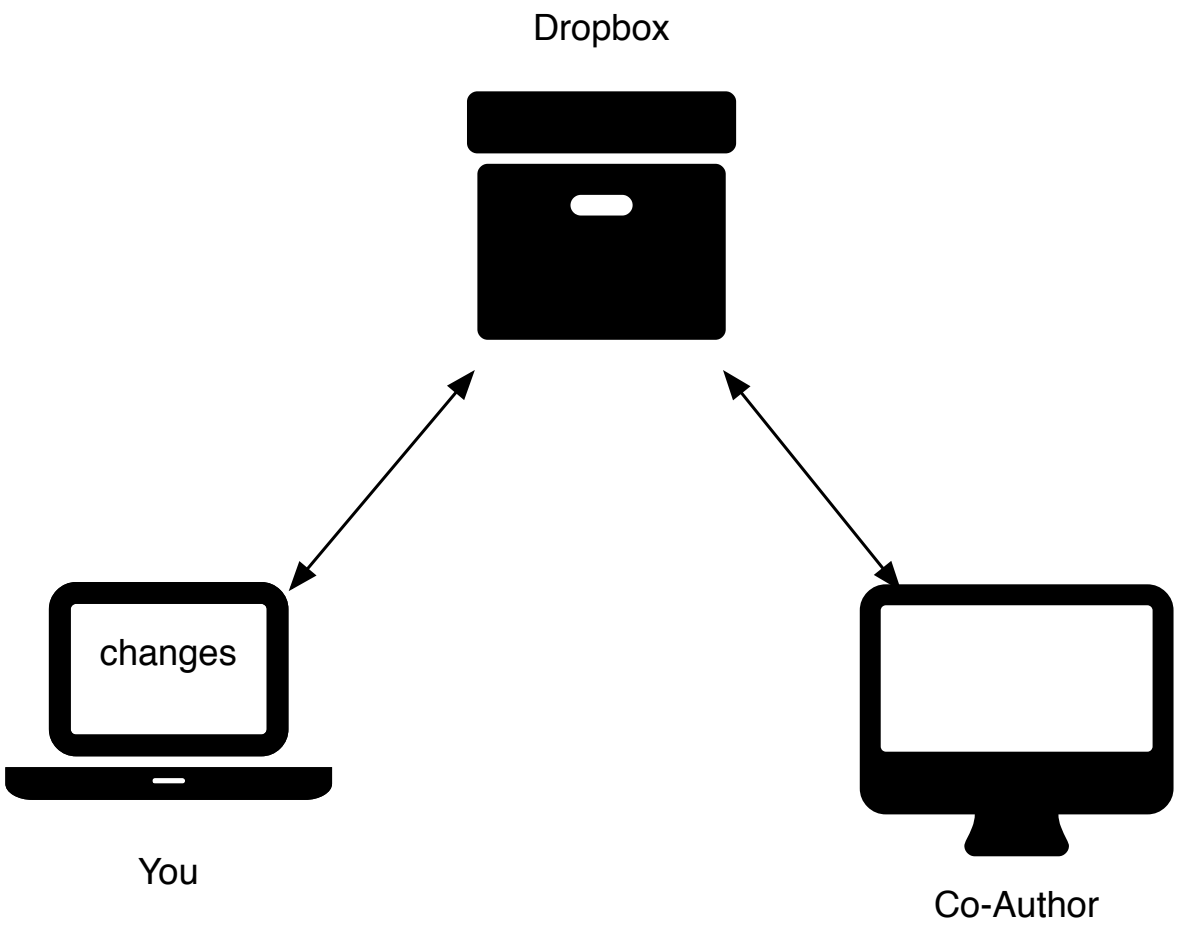
Distribution/Collaboration

- Git enables Collaboration – it is a distributed system.
  - Contrast to Dropbox

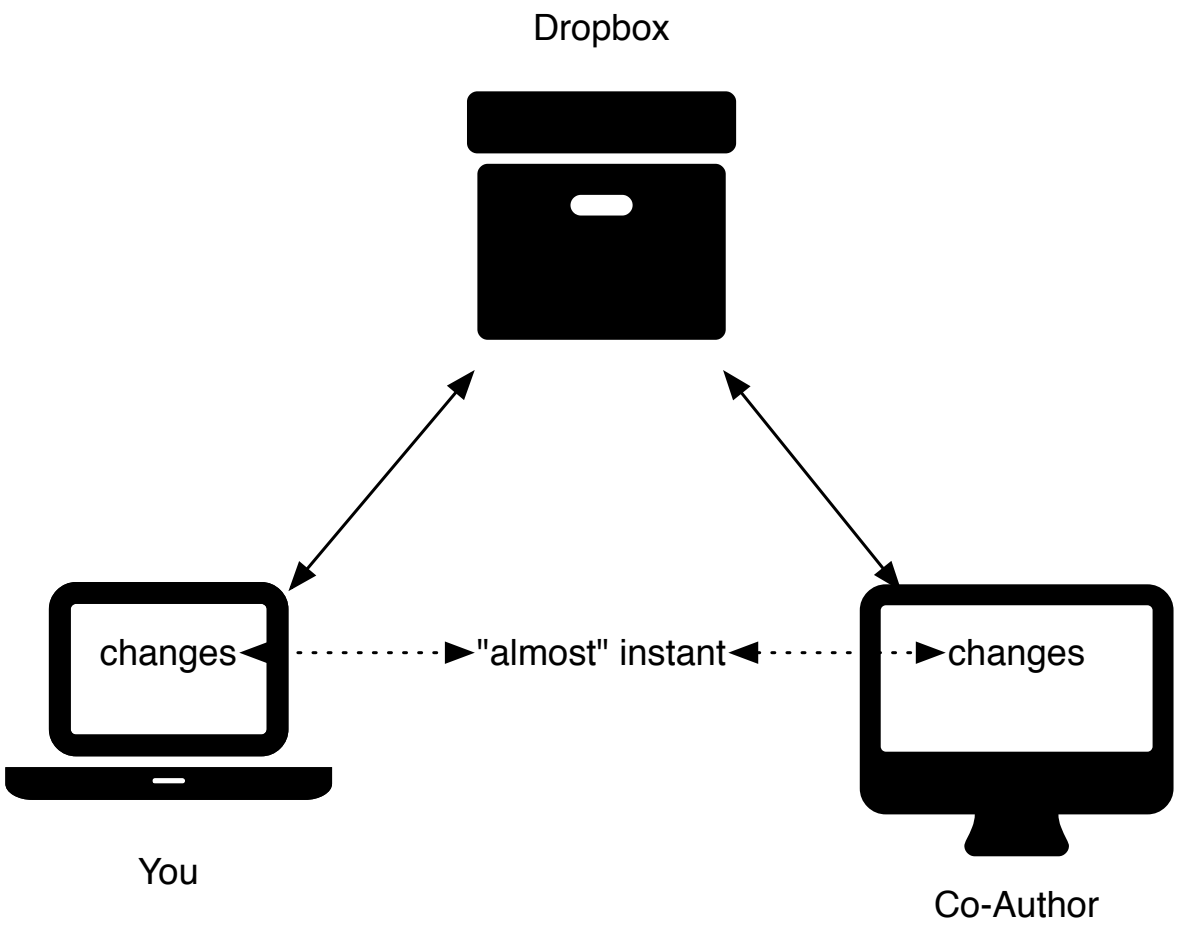
Distribution/Collaboration



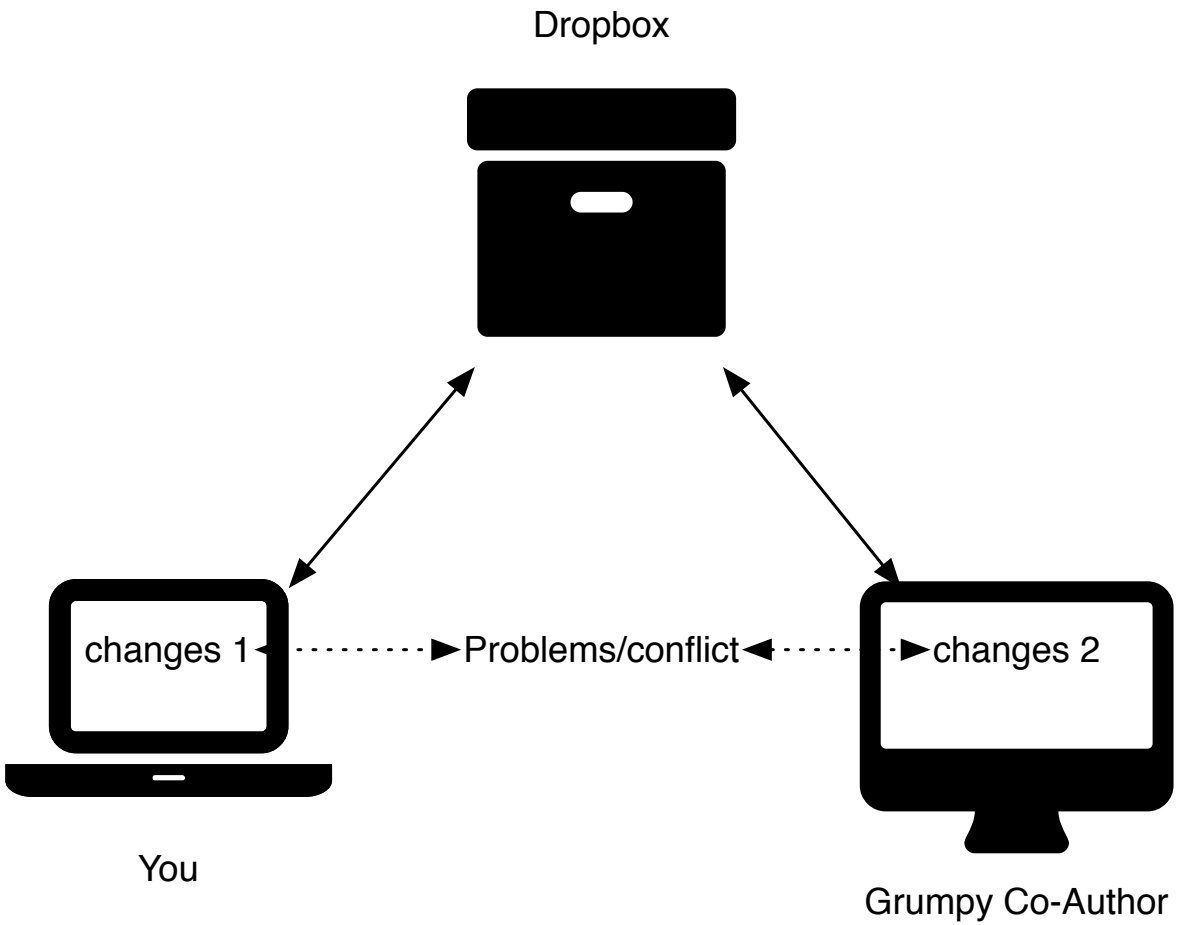
Distribution/Collaboration



Distribution/Collaboration



Distribution/Collaboration

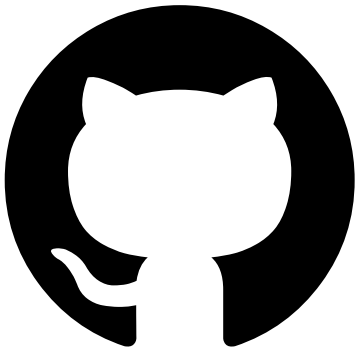


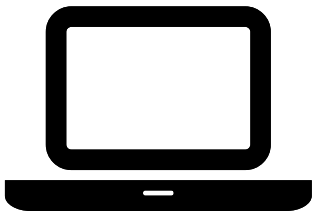
Distribution/Collaboration

- Git enables Collaboration – it is a distributed system.
  - Download repo to local computer
  - Make changes and commit
  - Push changes to server when ready
  - Pull changes from server when ready

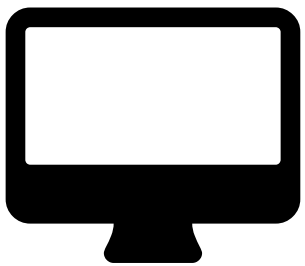
Distribution/Collaboration

Github
(or other service -- bitbucket, aws, etc.)



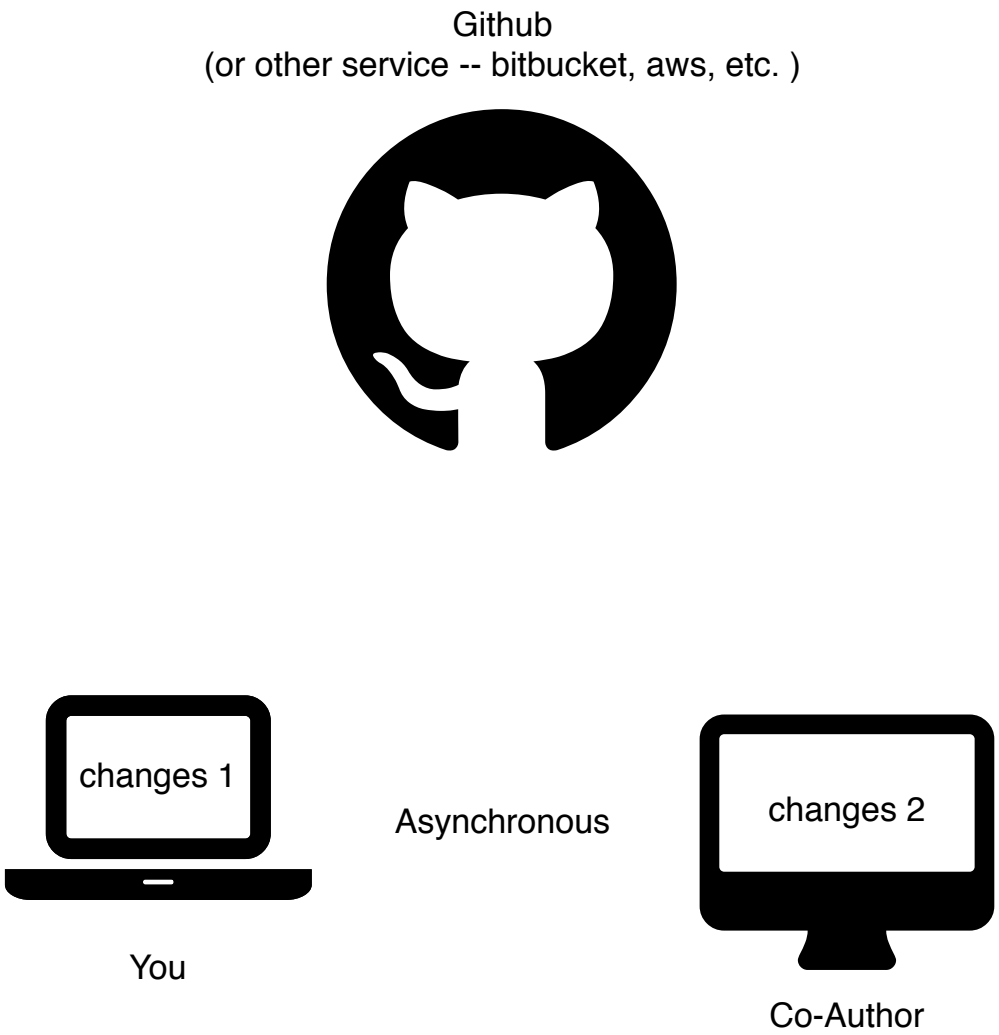


You

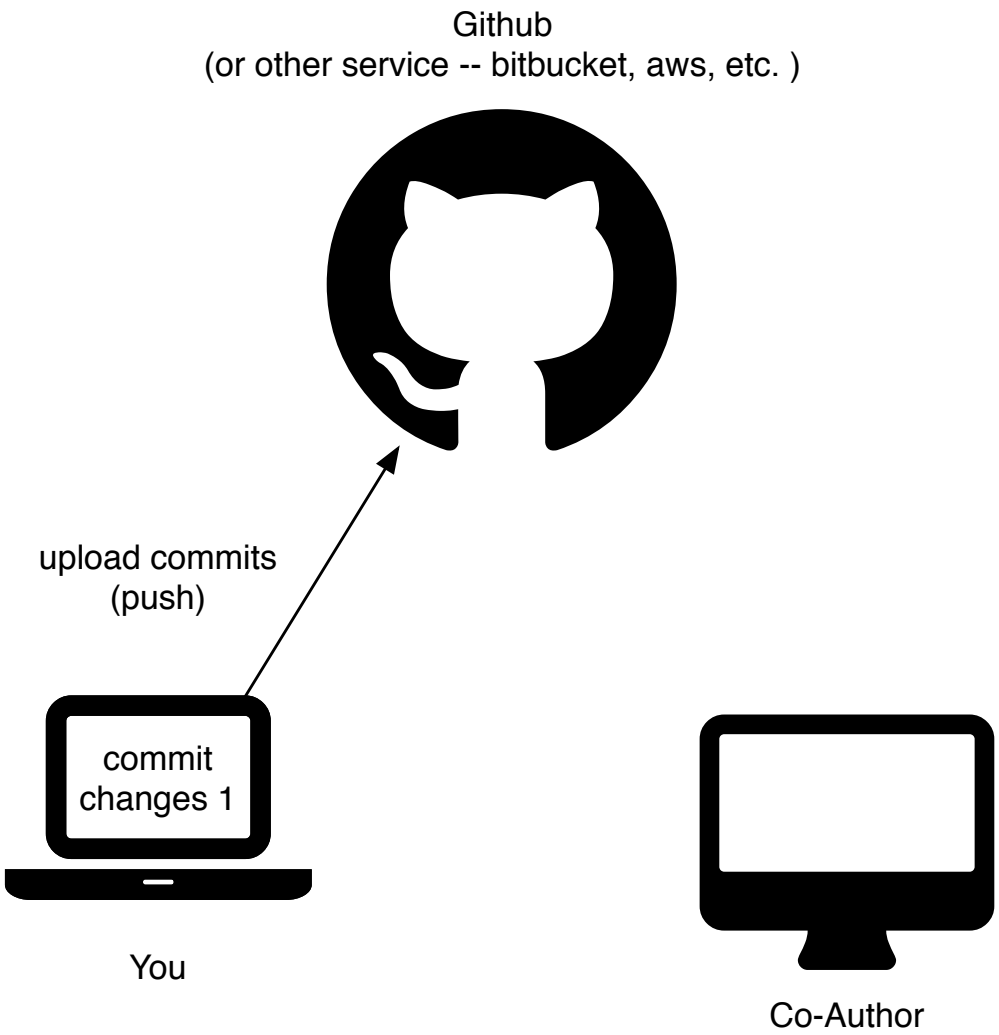


Co-Author

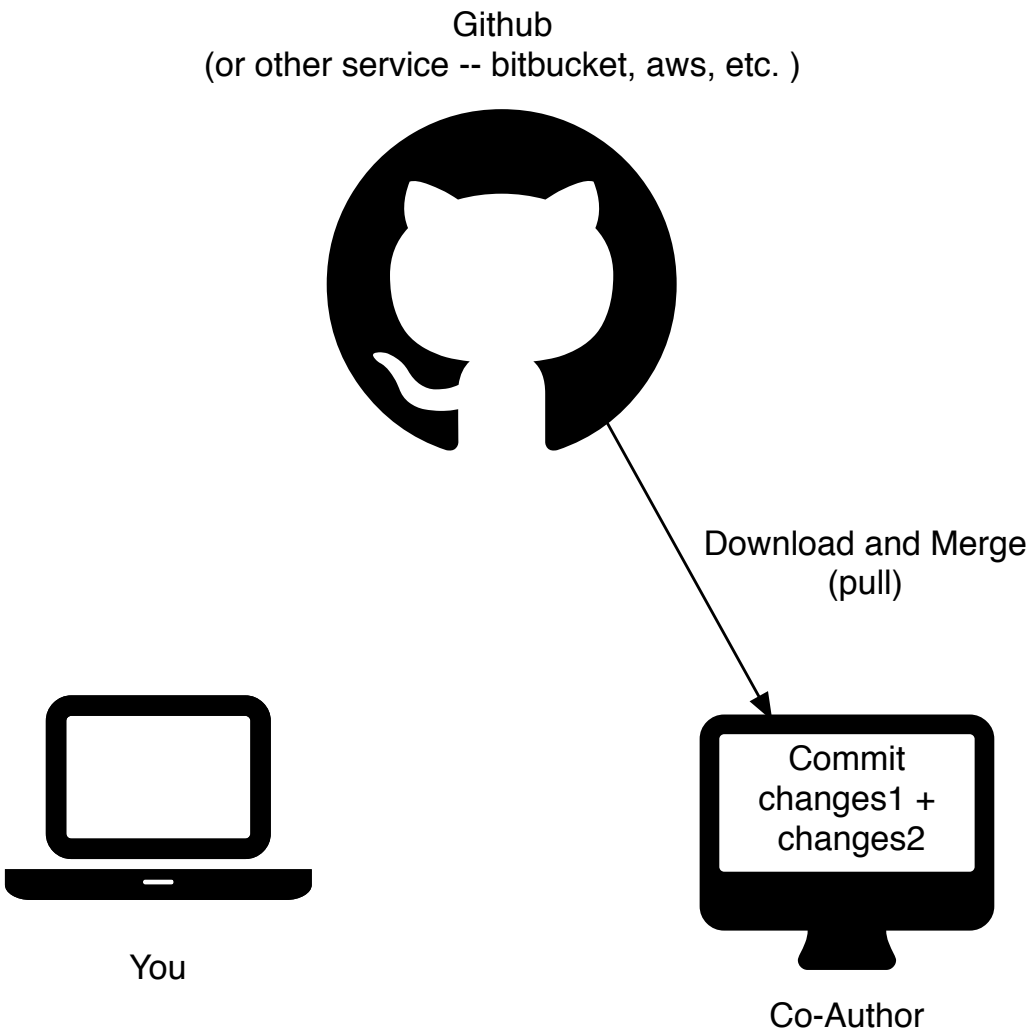
Distribution/Collaboration



Distribution/Collaboration



Distribution/Collaboration



Other things Git does

- Roll-back to previous versions
- Branch development/management
- Integration in to lots of software
- Best way to explore: start using git

Today

- Sourcetree
  - Setup Repo
  - Clone Repo
  - Checkout
  - Commit
  - Pull

Where to get help

- Easy help
  - Lots of places
  - Stackoverflow.com
  - Google
  - Github youtube channel
  - Sourcetree help
- Punching deck and interactive learning:
  - `try.github.io`
  - www.codeschool.com/courses/git-real
  - A great course at lynda.com www.lynda.com/Git-tutorials/Git-Essential-Training/100222-2.html
- Deep Dive
  - pro-git book by Scott Chacon and Ben Straub. – Free online `http://git-scm.com/book/en/v2`

Like Latex and R

- Totally Awesome
- Street Cred
- Learning Curve
    - Gets really fast/easy to use as time goes on

Things Git is bad at

- Tracking binary files – word files, images, etc. It will track them, but it's not ideal
  - This would be good use for Dropbox or something else
  - Not much benefit from tracking binary anyway

Merge Conflicts

- Git is good at fixing conflicts
- When it can't you need to fix merge conflicts
- Diff, resolve using 'mine'/'theirs'
  - Remember in Sourcetree: HEAD = 'mine'
  - External diff tools can help you cherry pick what to keep in a diff-merge
  - Meld on Windows(?)
  - On OS X, file-merge is included with xcode

What else can Git Do

- It works with rstudio
- Packages for atom and sublime
- You can use it to power a website

Two parts to a reproducible analysis: Blueprint and Machine

- Blueprint = code
  - Git helps with this
- Machine = computer (the software that runs the code)
  - Docker helps with this

The machine problem – more detail:

- Software versions change over time
  - R versions change
  - R packages change
  - Python, Ruby etc. Change even moreso
- Not every machine has the same configuration
  - Mac/Windows compatability problems – more common than you expect
    - ex. parallelized code

That's actually sort of a hard question to answer

- Virtualization software, but not totally
- But sort of if on pc/mac (needs to run in vbox – but still snappy)
- Upshot: Docker makes sure our code runs the same way every time, on any machine (with docker)

For Our Purposes

- Like Git for the computing machinery
  - Stable, consistent
  - Trackable over time
  - Sharable (dockerhub)

Use Cases (greater detail)

- Code critically depends on a package that is prone to changes
  - ggplot2 is a good example
- You use packages/etc that your co-author doesn't use
  - ex. python packages if you're scraping web
- You are developing package/software and you want to provide a demo environment
- Cloud-based HPC (let's talk if you're doing that)

At the moment...

- Docker is easy to use, but requires terminal/shell commands primarily
  - Kitematic is limited gui – can use to pull/run images, not make them

Docker, Images Containers

- Docker is a software platform (like Git)

Docker, Images Containers

- Docker is a software platform (like Git)
- Image $\approx$ defined computing environment (minimum: an os)

Docker, Images Containers

- Docker is a software platform (like Git)
- Image $\approx$ defined computing environment (minimum: an os)
  - Static

Docker, Images Containers

- Docker is a software platform (like Git)
- Image $\approx$ defined computing environment (minimum: an os)
    - Static
- Container: an instantiation of an Image

Docker, Images Containers

- Docker is a software platform (like Git)
- Image $\approx$ defined computing environment (minimum: an os)
  - Static
- Container: an instantiation of an Image
  - ephemeral, dynamic

Docker, Images Containers

- Docker is a software platform (like Git)
- Image $\approx$ defined computing environment (minimum: an os)
  - Static
- Container: an instantiation of an Image
  - ephemeral, dynamic
- Docker Runs Containers from Images

Docker, Images Containers

- Docker is a software platform (like Git)
- Image $\approx$ defined computing environment (minimum: an os)
  - Static
- Container: an instantiation of an Image
  - ephemeral, dynamic
- Docker Runs Containers from Images
- Analogy: USB drive

Dockerfiles

- Images are built from Dockerfiles
- Dockerfiles = base image (e.g. an os) + additional setup commands
- Example setup commands:
  - Add a directory or file to the image
  - install R
- Easy to build on prior images
  - Just pick image and specify it as base

Basic workflow:

- Write dockerfile Build image run containers

Basic Dockerfile

- Title these `Dockerfile` with no extension

```
FROM r-base
VOLUME /data
ADD ["data", "/data"]
ENTRYPOINT ["R","--no-save"]
```

- Full doc: `https://docs.docker.com/reference/builder`

Build Dockerfile To Image

- In terminal (in folder with Dockerfile):
  1. make sure docker machine is running: `docker-machine start default`
  2. make sure we are setup to use docker:`eval $(docker-machine env default)`
  3. Tell Docker to build: `docker build -t trcook/workshop_test .`
     - the `-t .../...` tells docker what to call the image internally. First part is username, don't use trcook – that's my name
     - the `.` at the end tells docker to look for `Dockerfile` in the current directory

Run docker from Image

- In terminal:
  - `docker run -it --rm trcook/workshop_test`
  - `-it` tells docker we want to interact with the container
  - `--rm` tells docker to remove the container after we are done (delete from memory) – Image will still remain

Expanded usage

- Can use this basic process for any project to create long-term reproduction image
- Caveat: will need to install required R packages through the Dockerfile

Install R packages:

```
FROM r-base
RUN Rscript -e "install.packages('pkg.name')"
RUN Rscript -e "m<-c('pkg1','pkg2','pkg3');install.packages
VOLUME /data
ADD ["data", "/data"]
ENTRYPOINT ["R","--no-save"]
```

Copy RUN for each package

Alternative 1:

```
FROM r-base
RUN Rscript -e "install.packages('pkg.name')"
RUN Rscript -e "m<-c('pkg1','pkg2','pkg3');install.packages
VOLUME /data
ADD ["data", "/data"]
ENTRYPOINT ["R","--no-save"]
```

Store packages in m

Alternative 2:

rsetup.R

```
m<-c('pkg1','pkg2')
install.packages(m)
```

```
FROM r-base
ADD ["rsetup.R", "/rsetup.R"]
RUN Rscript /rsetup.R
VOLUME /data
ADD ["data", "/data"]
ENTRYPOINT ["R","--no-save"]
```

Add and run seperate R-setup file

Push Images to Dockerhub

- hub.docker.com
  - repository for docker images – like github
  - `docker push trcook/workshop_test`
    - Requires you setup hub.docker.com acct first
  - Probably faster to run automated build

Automated Builds

- Docker and git play nice together
  - From hub.docker.com, point new build at git directory
  - Will build every time with push
  - Alternatively, will build once and stay static

Basics

- docs.docker.com
- A pretty good videofrom Learncode.academy
- Again – stack overflow is helpful here
- Me

- The end

- The end?