Docker, Git(hub)

MAS Departmental Disclaimers:

For students trying to take or audit from outside the MAS program.

Taking or auditing 400 courses is simply is not permitted because this is a self-supporting program. Sorry, unfortunately, you will NOT be able to take any of the 400 level Stats courses.

There are NO exceptions that can be made by the department. These classes were designed specifically for students who applied directly to the program.

The students of this program are also not allowed to audit or enroll in classes outside of the program as it was created for working professionals.

If you would like to apply for the program, you are welcome to do so: https://master.stat.ucla.edu/admissions/

Information is found here: https://master.stat.ucla.edu/ And here: https://master.stat.ucla.edu/faq/

A reminder…

class communications

Slack Channel
uclastat418-class

Last time we saw an introduction to docker

today we'll make use of it while also hearing a bit more about what it is.

throughout the rest of course (in most weeks) we will continue to make use of docker environments to incrementally learn about its features

Docker: what is it?

a software to build and run containers with a container being a single instance of an image

an image is a unit of software that allows one to package up code and all its dependencies

Docker: why is it important?

different software development stacks require different environments; especially dependent on task

classical solutions entail virtual environments (anaconda, python virtualenv) or virtual machines (vagrant, vmware)

Docker: so why do data science with it?

it is platform agnostic; as long as a machine has docker we can use the same environment (perfect in a classroom setting where we all have different machines; perfect in a work setting where we might have different machines and certainly different versions)

can share version-controlled environments akin to GitHub for docker images

containers are ephemeral; if you mess up you can start over easily

build on top of other peoples work

ready-made workflow for deployment through Kubernetes or similiar

More on Containerization

Containerization involves packaging software code along with the essential operating system libraries and dependencies it needs to run, creating a self-contained and lightweight executable called a container. This container is designed to run consistently across any infrastructure, eliminating the inconsistencies often encountered when moving applications between different computing environments.

Compared to traditional virtual machines, containers offer greater portability and are more resource-efficient, making them the standard building blocks for modern cloud-native applications. This technology empowers developers to build and deploy applications more rapidly and securely by bundling all necessary components, ensuring they run reliably regardless of the underlying platform or cloud environment – effectively enabling a "write once, run anywhere" approach.ready-made workflow for deployment through Kubernetes or similiar

While the underlying concept has existed for decades, the widespread adoption of containerization was significantly accelerated by the emergence of Docker in 2013. Docker provided user-friendly tools and a standardized way to package applications, leading organizations to increasingly leverage containers for developing new applications and modernizing existing ones for the cloud, benefiting from their lightweight nature, faster startup times, and the ability to achieve higher server utilization and reduced costs.

How Containerization Works

Encapsulating Applications - Containers package an application as one executable. This bundle includes the application code, configuration files, libraries, and dependencies needed to run.

Sharing the Operating System - Unlike VMs, containers don't include a full OS. Instead, they rely on a container runtime (like Docker) installed on the host OS. This runtime allows containers to share the host's operating system.

Efficiency Through Sharing and Isolation - Containers can share common resources like binaries and libraries. This sharing reduces overhead and makes containers smaller and faster to start than VMs. The isolation of each container also enhances security, preventing issues in one container from affecting others or the host system.

Achieving Portability and Consistency - By abstracting away from the host OS, containerized applications become highly portable. They can run uniformly across various platforms and clouds, from desktops to VMs, and across different operating systems or server environments.

Enabling Faster and Secure Development - Containerization speeds up and secures application development and deployment. This applies to both traditional monolithic applications and modern microservices architectures. Developers can build new cloud applications as containerized microservices or repackage existing applications for better resource use.

Virtualization vs Containerization

Both virtualization and containerization improve computing efficiency by enabling multiple software instances to run on a single physical machine.

Virtualization: Bundling the OS

- Utilizes a hypervisor to separate the OS and applications from the underlying hardware.
- Allows running multiple operating systems (e.g., Windows and Linux) and applications simultaneously on one host.
- Each VM includes its own operating system, application files, libraries, and dependencies.
- Achieves cost savings by consolidating multiple workloads onto fewer physical servers.

Containerization: Sharing the OS

- Packages applications with their necessary dependencies into a single executable.
- Does not include a separate operating system. Instead, containers share the host OS through a container runtime.
- Often described as "lightweight" due to OS sharing and smaller size.
- Offers faster startup times and allows more applications to run on the same hardware compared to VMs.
- Leads to even higher server efficiencies and reduced costs.

Docker: running Rstudio

run the following

```
docker run -e PASSWORD='class' --rm -p 8787:8787 rocker/rstudio
```

and then visit in any browser

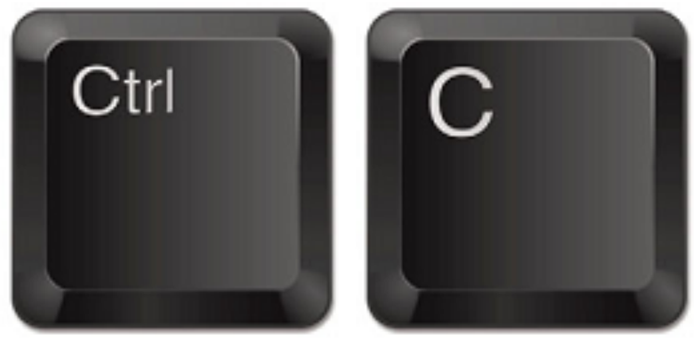
`http://localhost:8787` or `http://<your-ip-address>:8787`

finally login with the following credentials

Username: `rstudio` Password: `<what you used above>`

Docker: running Rstudio

to exit



Copy (CTRL-C)

We will see another way to exit in a moment

Docker: running Rstudio with mounting local volume (connect to your computer)

cd to desired directory, then use for Mac

```
docker run -d --rm -e PASSWORD=class -p 8787:8787 -v `pwd`:/home/rstudio/Documents
rocker/rstudio
```

chdir to desired directory in Windows command line

```
docker run -d --rm -e PASSWORD=class -p 8787:8787 -v %cd%:/home/rstudio/Documents
rocker/rstudio
```

cd to desired directory in Windows powershell

```
docker run -d --rm -e PASSWORD=class -p 8787:8787 -v ${PWD}:/home/rstudio/Documents
rocker/rstudio
```

Again visit in your browser port 8787 either at local host or <your-ip-address> and login with your credentials

Docker: a few other commands

Now to exit we will need to take a look at what containers are running to find the container id or name

```
docker container ls
```

```
docker kill <container_name>
```

And finally, to see what docker images we have available

```
docker images
```

Docker: a few other commands

And finally, to see what docker images exist on our machine

```
docker images
```

If you'd like to remove any of these images

```
docker rmi <image id>
```

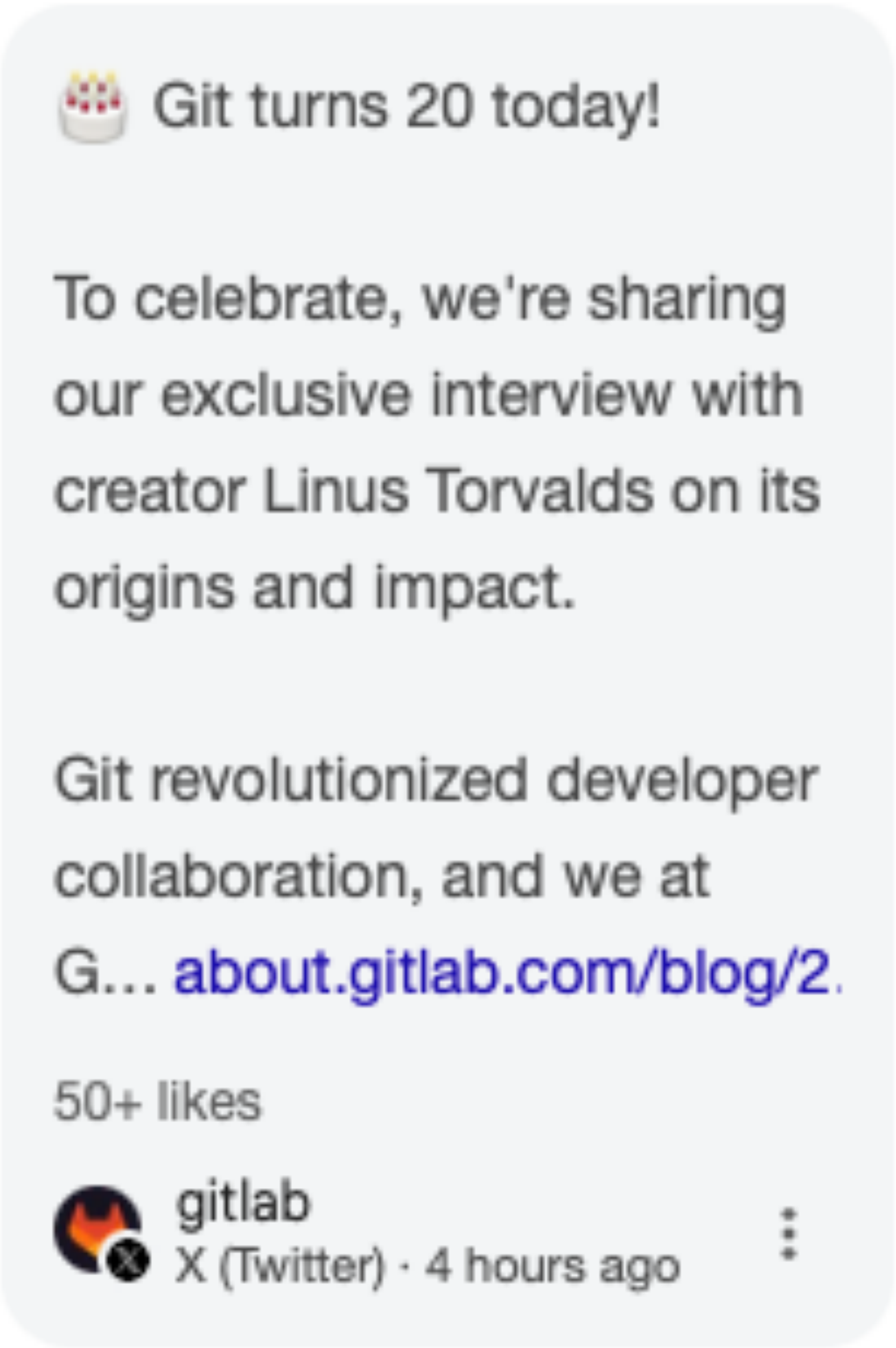
Git

Git: what is it?

a version control system to track the history of changes as people and teams collaborate on code projects together

Git manages the evolution of a set of files (or in our case a data analysis or model development) in a repository

this provides a transparent history of changes, who made them and how each



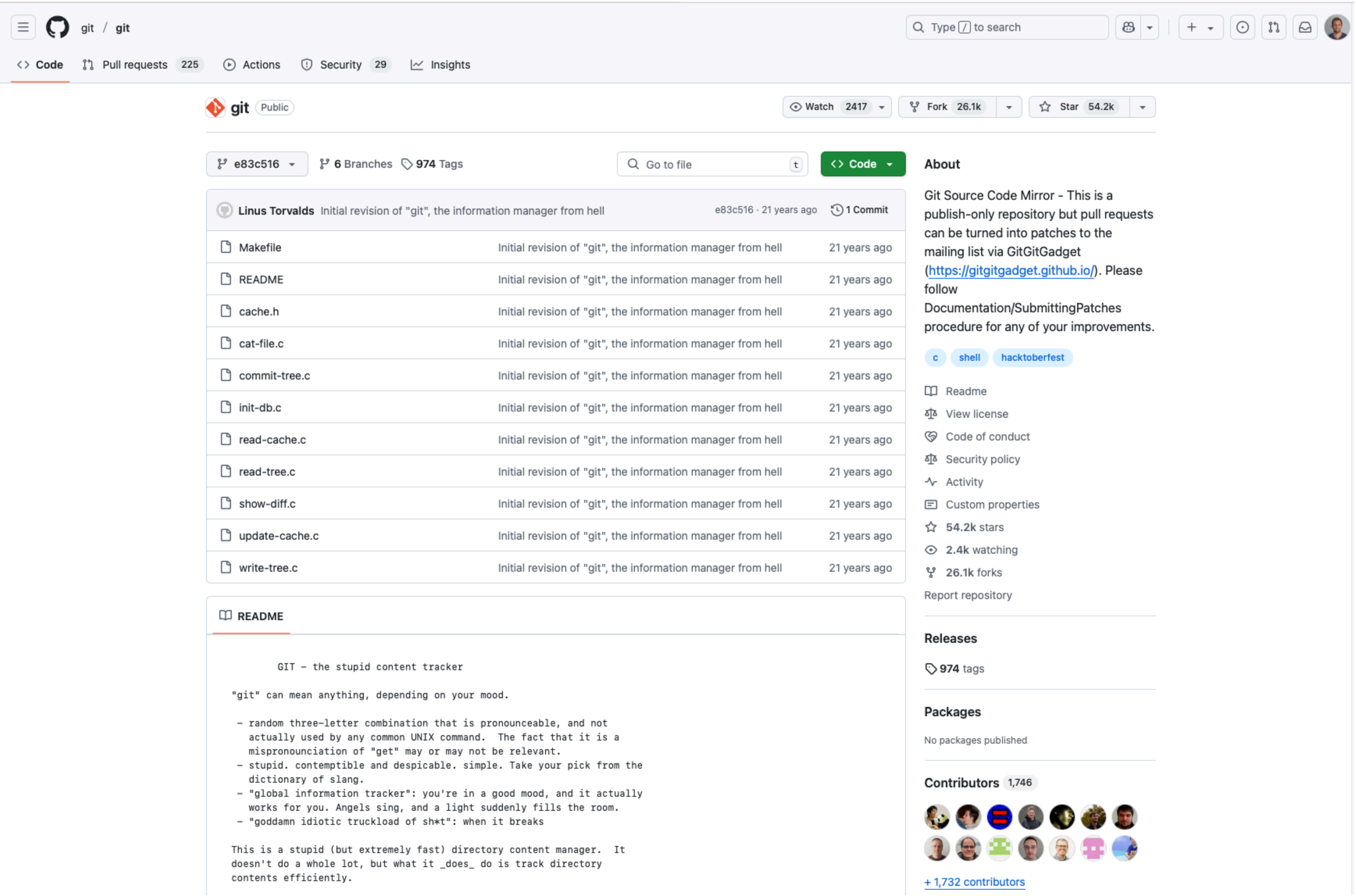
04/07/2025

A history

In 2005, Linus Torvalds was developing Linux when BitKeeper (proprietary source-control management) revoked their free license. Torvalds set 4 goals for his development needs for a SCM and when he couldn't find anything sufficient he decided to create his own. Development began April 3 and was released April 7.

"I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'Git'"."

('git' is British slang for "pig headed, think they are always correct, argumentative").



https://github.com/git/git/tree/e83c5163316f89bfbde7d9ab23ca2e25604af290

Git

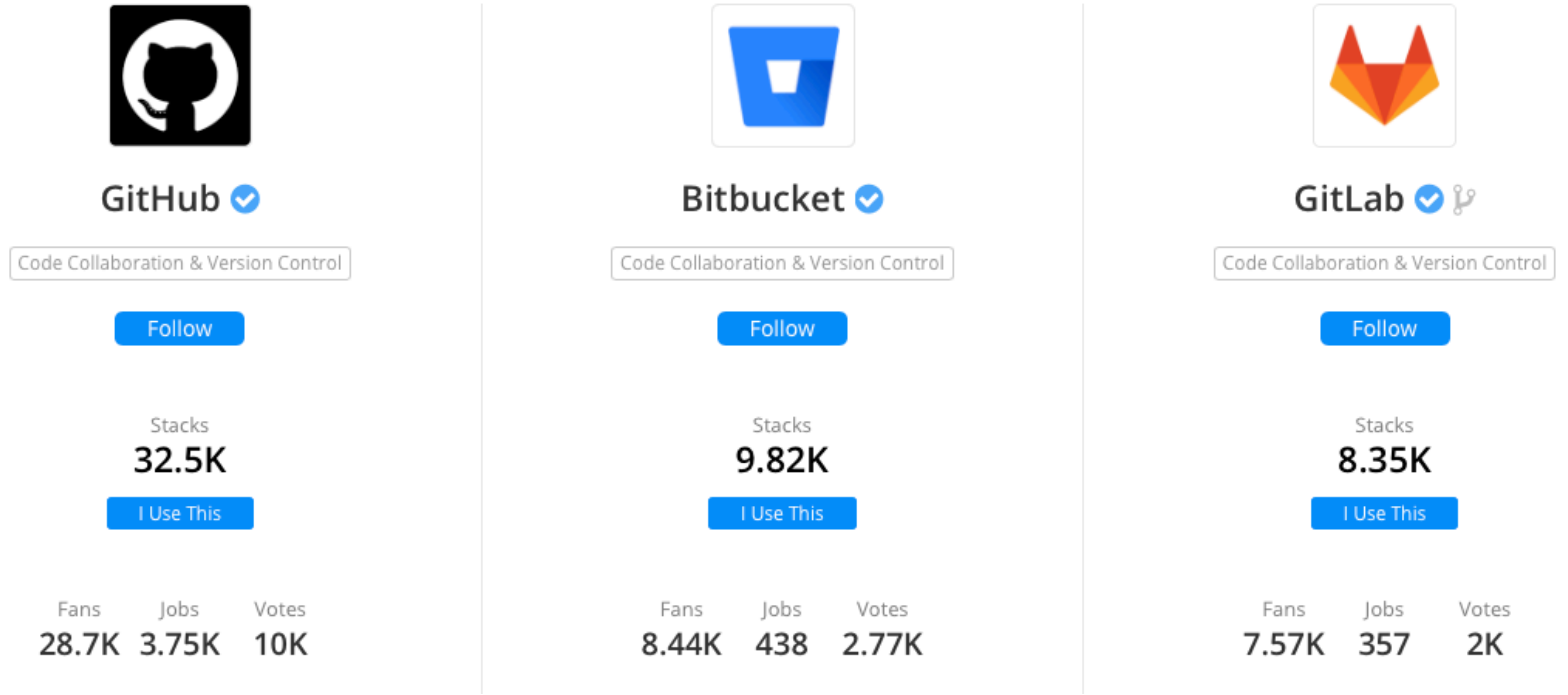
|< < Prev Random Next > >|

|< < Prev Random Next > >|

Permanent link to this comic: https://xkcd.com/1597/
Image URL (for hotlinking/embedding): https://imgs.xkcd.com/comics/git.png

Repository management services

Repository management services



Repository management services

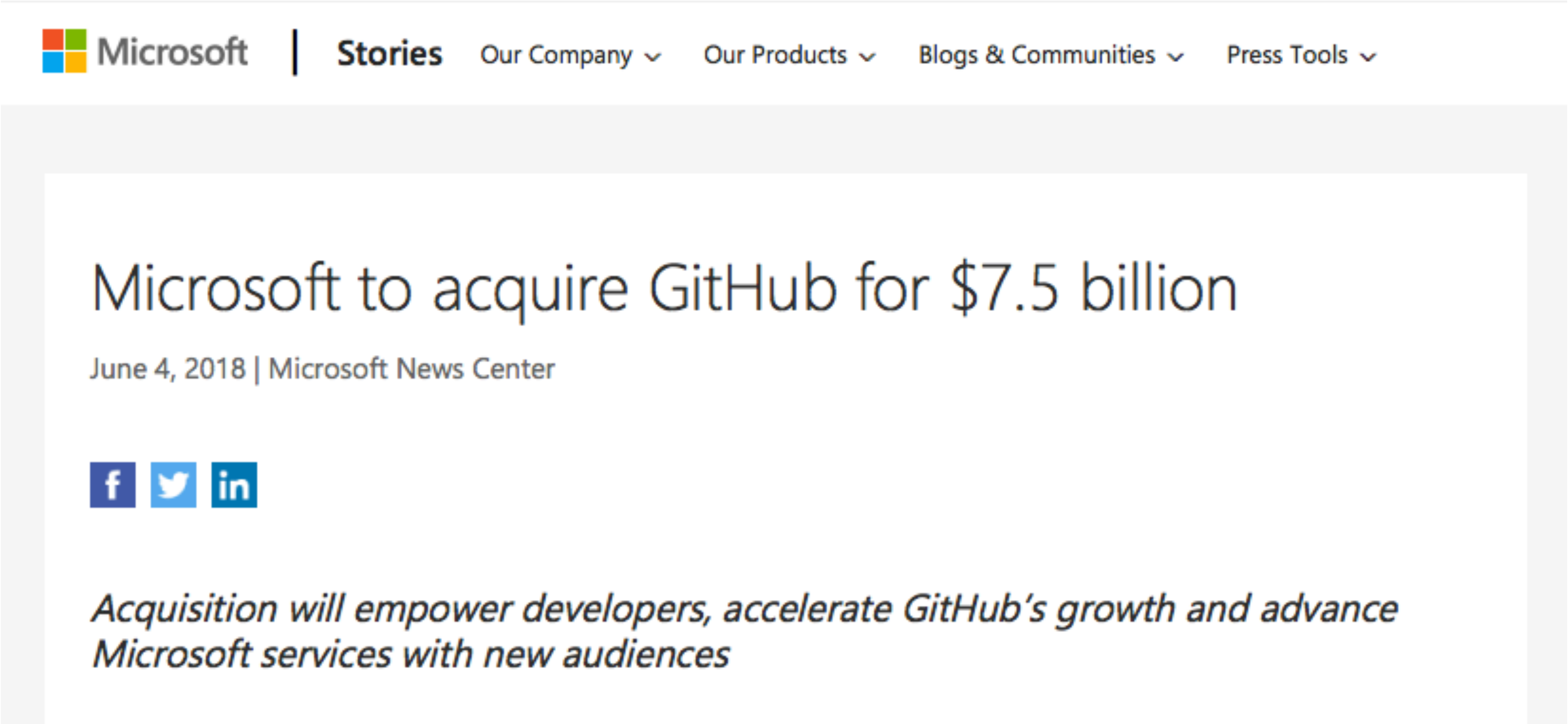
What companies use GitHub?	What companies use Bitbucket?	What companies use GitLab?
5837 companies on StackShare use GitHub	*2270 companies on StackShare use Bitbucket*	*1584 companies on StackShare use GitLab*
Airbnb	PayPal	Alibaba.com
Netflix	Salesforce	Coderus
Medium	Zillow	Webedia
Instacart	Starbucks	Ticketmaster
reddit	CircleCI	WebbyLab
Lyft	Tesla Motors	Edify
StackShare	Bitbucket	Electronic Arts
Shopify	Pandora	Freelancer.com
9GAG	Sellsuki	Citrix
Asana	Movielala	WILD

We’re going to use GitHub

http://github.com

Make an account if you don’t have one

Let it be known…


Microsoft | Stories Our Company Our Products Blogs & Communities Press Tools

Microsoft to acquire GitHub for $7.5 billion

June 4, 2018 | Microsoft News Center

Acquisition will empower developers, accelerate GitHub's growth and advance Microsoft services with new audiences

https://www.theverge.com/2018/6/18/17474284/microsoft-github-acquisition-developer-reaction

It’s not all fun and games. Actually it’s quite hard. Initially just working on solo repositories becomes fine but once collaborating you’ll start to run into merge conflicts (yikes)

Keep going!!

GitHub: what tool to use?

Use what makes you feel comfortable and efficient.

> *"I certainly know Git very well, and honestly think I'm far faster and more efficient in a Git GUI than I could possibly be in the command line – and I'm certainly not slow in the CLI."*
>
> —Dan Clarke, blogger and co-organizer of .Net Oxford group

> *"I sometimes encounter people who feel it's "better" to use command line Git, but for very ill-defined reasons. These people may feel like they should work in the shell, even if it leads to Git-avoidance, frequent mistakes, or limiting themselves to a small set of ~3 Git commands. This is counterproductive."*
>
> —Jenny Bryan, Software Engineer at RStudio & Adjunct Professor at the University of British Columbia

Both quotes take from https://blog.axosoft.com/git-gui-vs-cli/

GitHub GUIs

axosoft GitKraken

GitHub Desktop

SourceTree

There are a wide variety of choices for GUIs if you are interested.

GitHub: some (basic) guidelines

Personally, I start all my repositories remotely (on GitHub.com). I find it much easier to then pull to my local machine and to then make my first commit

This includes forking others' repos and then cloning to local

Create new branches to make edits (add new features) to the master code base

Commit early and often; typically don't push data (especially when large) and definitely not credentials

(these are all quite simple; you'll surely run into many issues and there are many more resources online that will help you, but don't be afraid to try some new things in this course to learn how it works)

GitHub: our class repository

Let's create a repo for you to use and push hw to my master repo. cd to the directory where you want to

Fork the class repo on GitHub. Clone your https version of the repo.

Now let's go to docker and do this in a (hopefully) more familiar environment…get an Rstudio container running with a local volume mounted. (if its not still running at the moment)

…will walk through as a class.

A bit more on reproducibility…

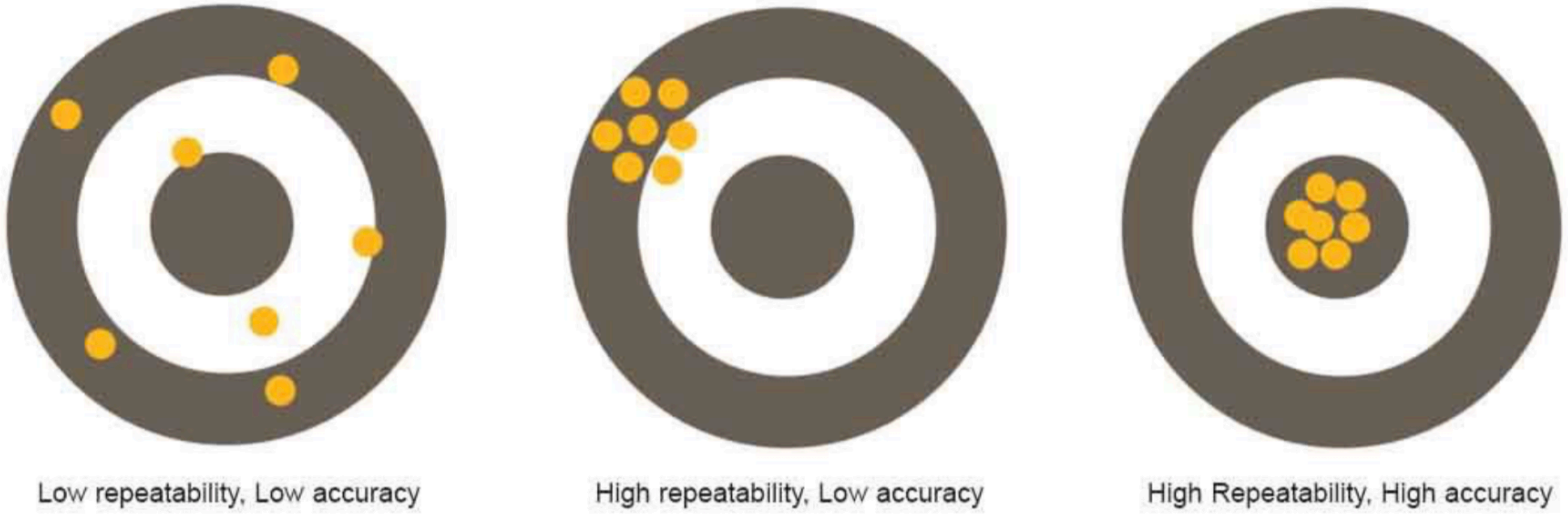
there is a bit of crisis taking place in regards to reproducibility as a whole in science fields. and this true of the data science field as well.

We've seen some of the tools to enable us to do this reproducible work, but what concepts should we be thinking of as we use them?

Repeatability

or test–retest reliability is the variation in measurements taken by a single person or instrument on the same item, under the same conditions, and in a short period of time. A less-than- perfect test–retest reliability causes test–retest variability.

Your systems must be repeatable and reliable, the same query on the same data should return the same results. Otherwise, none of this applies.



Reproducibility

an analysis is *reproducible* if there is a specific set of computational functions/ analyses (usually specified in terms of code) that exactly reproduce all of the numbers in a published paper from raw data. It is now recognized that a critical component of the scientific process is that data analyses can be reproduced.

Replicability

but just because a study is reproducible does not mean that it is *replicable*. Replicability is stronger than *reproducibility*. A study is only *replicable* if you perform the exact same experiment (at least) twice, collect data in the same way both times, perform the same data analysis, and arrive at the same conclusions.

Stodden's Taxonomy of Reproducibility

Computational

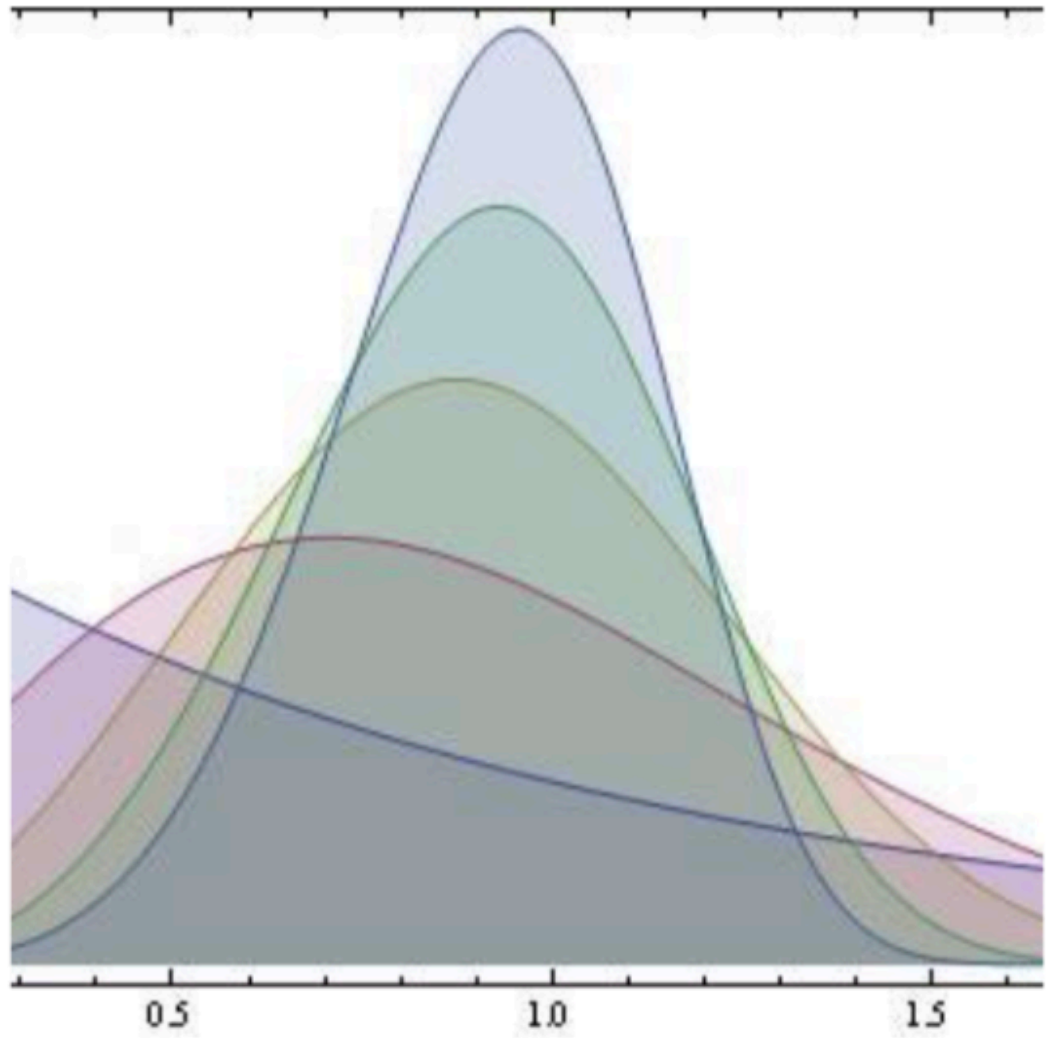
when detailed information is provided about code, software, hardware and implementation details.

Empirical

when detailed information is provided about code, software, hardware and implementation details.

Statistical

when detailed information is provided about code, software, hardware and implementation details.



Computational reproducibility should be…

Reviewable

The descriptions of the research methods can be independently assessed and the results judged credible

Replicable

tools are made available that would allow one to duplicate the results of the research, for example by running the authors' code to produce the plots shown in the publication.

Auditable

sufficient records (including data and software) have been archived so that the research can be defended later if necessary or differences between independent confirmations resolved. The archive might be private, as with traditional laboratory notebooks.

Confirmable

the main conclusions of the research, or the model in production can be attained independently without the use of software provided by the author.

Simple rules for reproducible computational research

track results - whenever a result may be of potential interest, keep track of how it was produced. as a minimum, you should at least record sufficient details on programs, parameters, and manual procedures to allow yourself, in a year or so, to approximately reproduce the results.

script everything - whenever possible, rely on the execution of programs instead of manual procedures to modify data. If manual operations cannot be avoided, you should as a minimum note down which data files were modified or moved, and for what purpose.

store program versions - In order to exactly reproduce a given result, it may be necessary to use programs in the exact versions used originally. As a minimum, you should note the exact names and versions of the main programs you use.

Simple rules for reproducible computational research

use version control - even the slightest change to a computer program can have large intended or unintended consequences. As a minimum, you should archive copies of your scripts from time to time, so that you keep a rough record of the various states the code has taken during development

store data & intermediate results - in principle, as long as the full process used to produce a given result is tracked, all intermediate data can also be regenerated. In practice, having easily accessible intermediate results may be of great value. As a minimum, archive any intermediate result files that are produced when running an analysis

set a random number seed- many analyses and predictions include some element of randomness, meaning the same program will typically give slightly different results every time it is executed. As a minimum, note which analysis steps involve randomness, so that a level of discrepancy can be anticipated when reproducing the results.

Simple rules for reproducible computational research

store data viz inputs - from the time a figure is first generated to it being part of an analysis. It is critical to store the data and process that generated it. As a minimum, one should note which data formed the basis of a given plot and how this data could be reconstructed.

allow levels of analysis - in order to validate and fully understand the main result, it is oLen useful to inspect the detailed values underlying the summaries. Make those fluid and explorable, as a minimum at least once generate, inspect, and validate the detailed values underlying the summaries.

generate text programmatically- there is nothing quite as embarrassing as having a disagreement between your analytical narrative in a writeup and having the tables and figures disagree. Connect them so that there is no change of disagreement.

.