

version control : git + GitHub

Marco Morales

marco.morales@columbia.edu

Nana Yaw Essuman

ne2388@columbia.edu

GR5069: Applied Data Science
for Social Scientists

Spring 2026
Columbia University

a quick reminder...



no open laptops



no cellphones

A large collection of colorful plastic cleaning bottles, including spray bottles and jugs, in shades of yellow, orange, and red, arranged in rows.

housekeeping

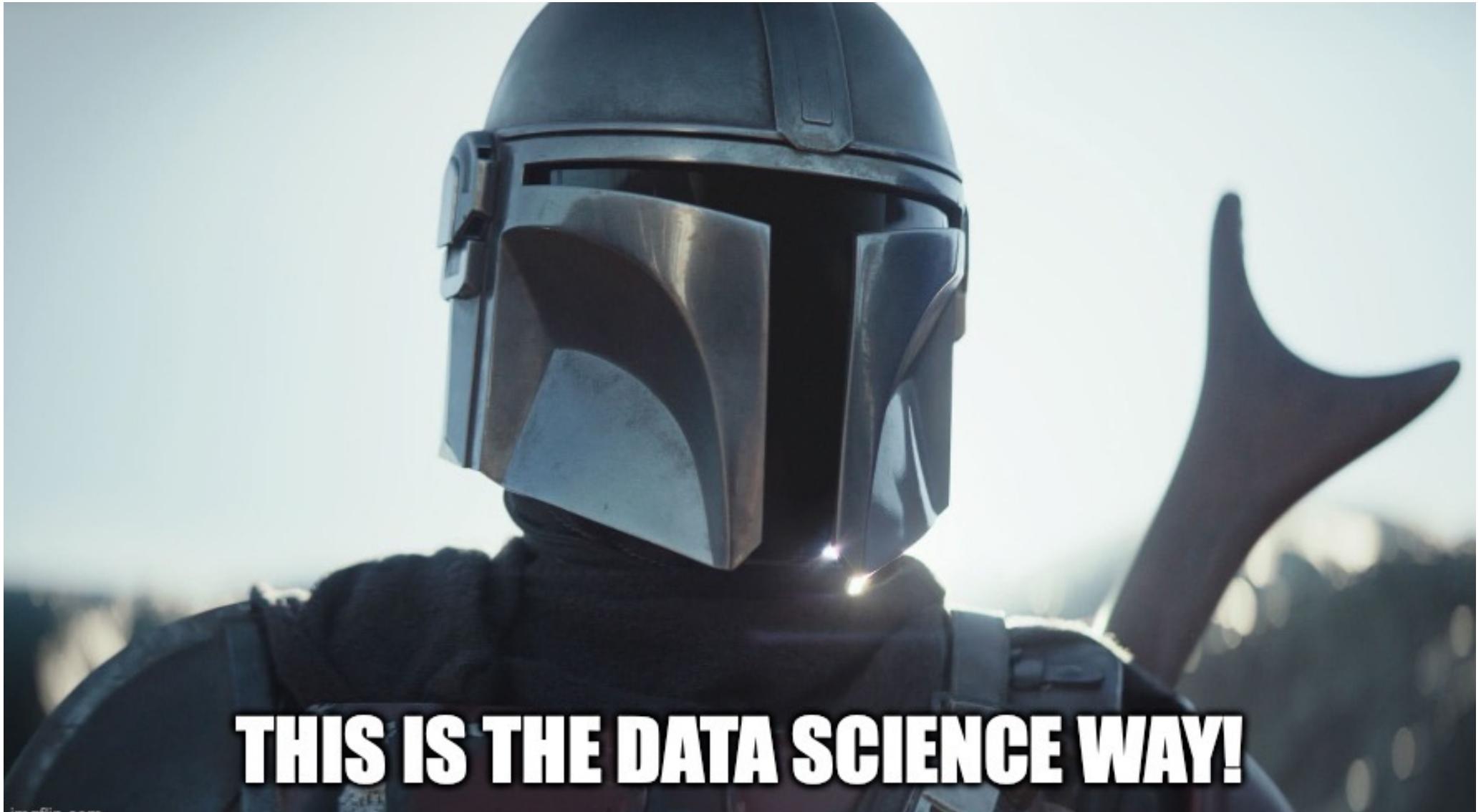
housekeeping!

- invites to the course's **Slack** workspace have been sent out to your Columbia email. **Accept invite to Slack**
- please **submit** your **GitHub handle** in the Courseworks assignment.
- you'll receive invites to AWS and Databricks as needed. **Follow the instructions to accept the invites.**



slight detour

recap: collaborate, collaborate, collaborate!



THIS IS THE DATA SCIENCE WAY!

behold messaging systems!

- your team will likely use a **messaging system** to collaborate
- important that you get used to it!
- we'll use Slack for this class as a real-world simulation



remember the 15 min rule?

- first, **try to figure it out** on your own
- if you haven't figured it out **after 15 minutes**, ask your teammates / classmates for **help**
- never helpful to go down rabbit holes on your own!
- likely that someone else has **faced and solved your question/problem** and may help you with it
- post your question in the **appropriate Slack channel!**

a bit of Slack etiquette

- mention people (i.e. **@marco-morales**) when speaking to them directly on a channel
 - people will not be notified unless you mention them
- use **@channel** and **@here** with care
 - **@here** notifies all people currently active in the channel
 - **@channel** notifies all members of the channel
- be mindful of other people's time and schedules (and time zone differences!)

a few useful Slack gimmicks!

- Slack works on **Markdown**, so it's simple to format the text of your messages
- easy to share snippets of **code, text, data**
- can **edit messages after sending** them (nice alternative for documenting)
- **integrations** with other apps

A black and white photograph of a checkered racing flag flying diagonally from the bottom right towards the top left. The flag has a white border and is set against a background of a cloudy, overcast sky.

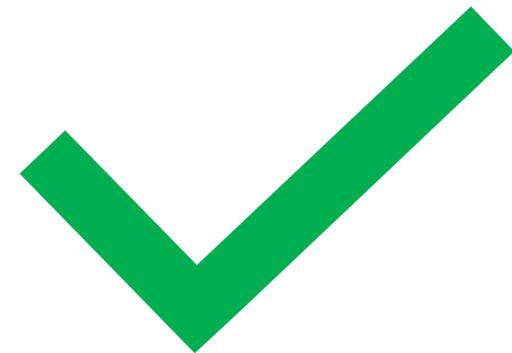
let's get started!

recap: what does a Data Scientist do?

Instagram

vs

reality



recap: what does a Data Scientist do?



problem: a statement without (an appropriate) solution

solution: a Data Product that (effectively) mitigates a problem

recap: what does a Data Scientist do?

what



frontends for automated data summarization and visualization



science-backed answers to business questions
(explanations, scenarios, projections, causes)



stand-alone algorithmic outputs that integrate to business processes



end-to-end proprietary applications developed to fulfill a business objective

who



data engineer data analyst



data engineer data scientist



data engineer ML engineer data scientist



data engineer ML engineer data scientist data analyst project manager

where



solutions architecture



computing architecture

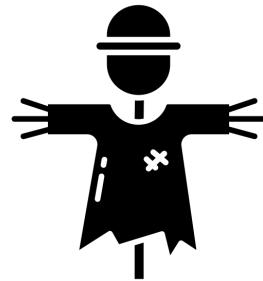


computing architecture

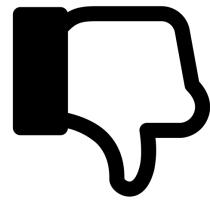


1

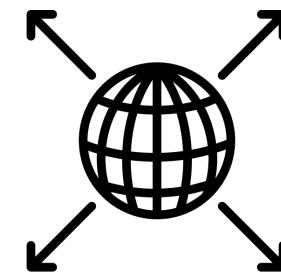
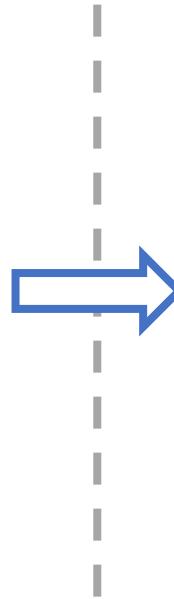
recap: iteration to build Data Products



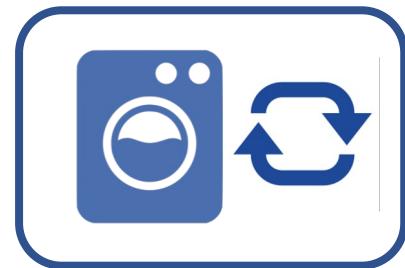
start small
(MVP)



fail fast

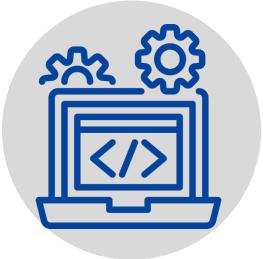


scale up



iterate

working environments to build Data Products



dev

development

- write **code**
- fix **bugs**
- perform **tests**



QA

quality assurance
/ testing

- find **issues**
- reveal **problems**
- run **test cases**
- ensure **things work as expected**



stg

staging

- mirror **prod environment**
- ensure **prod will not “break”**

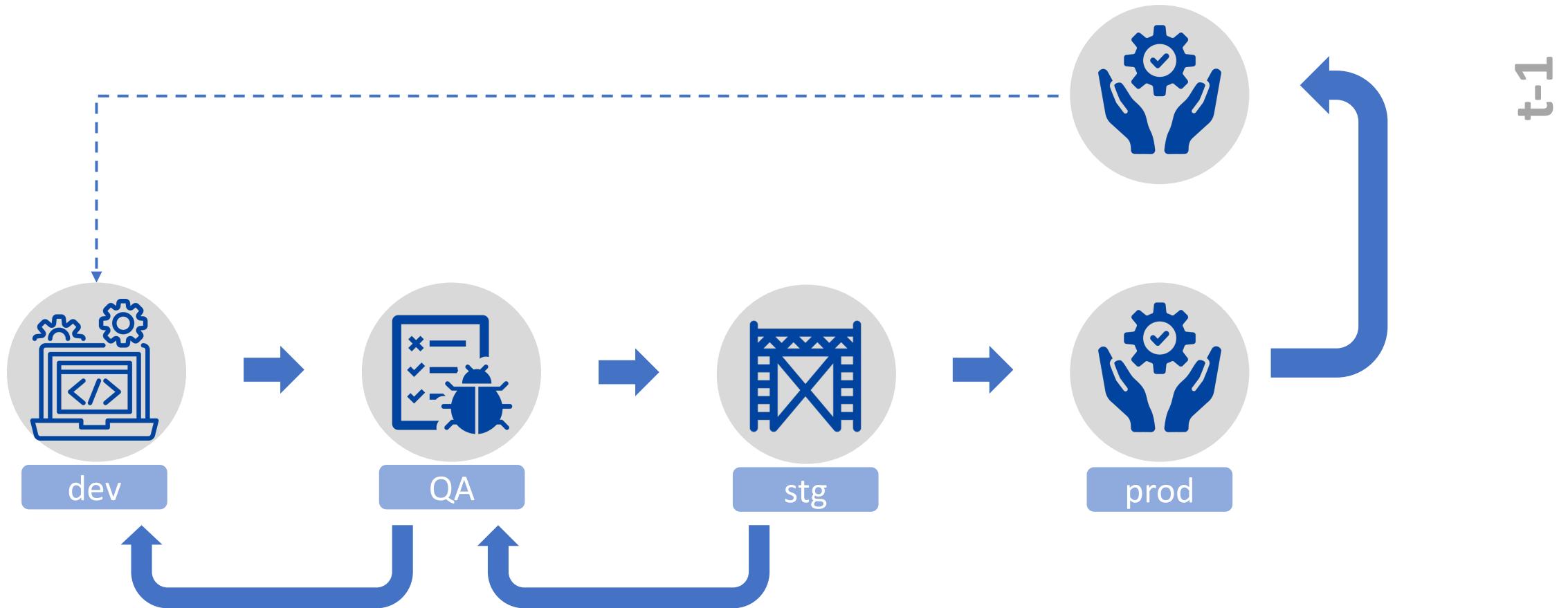


prod

production

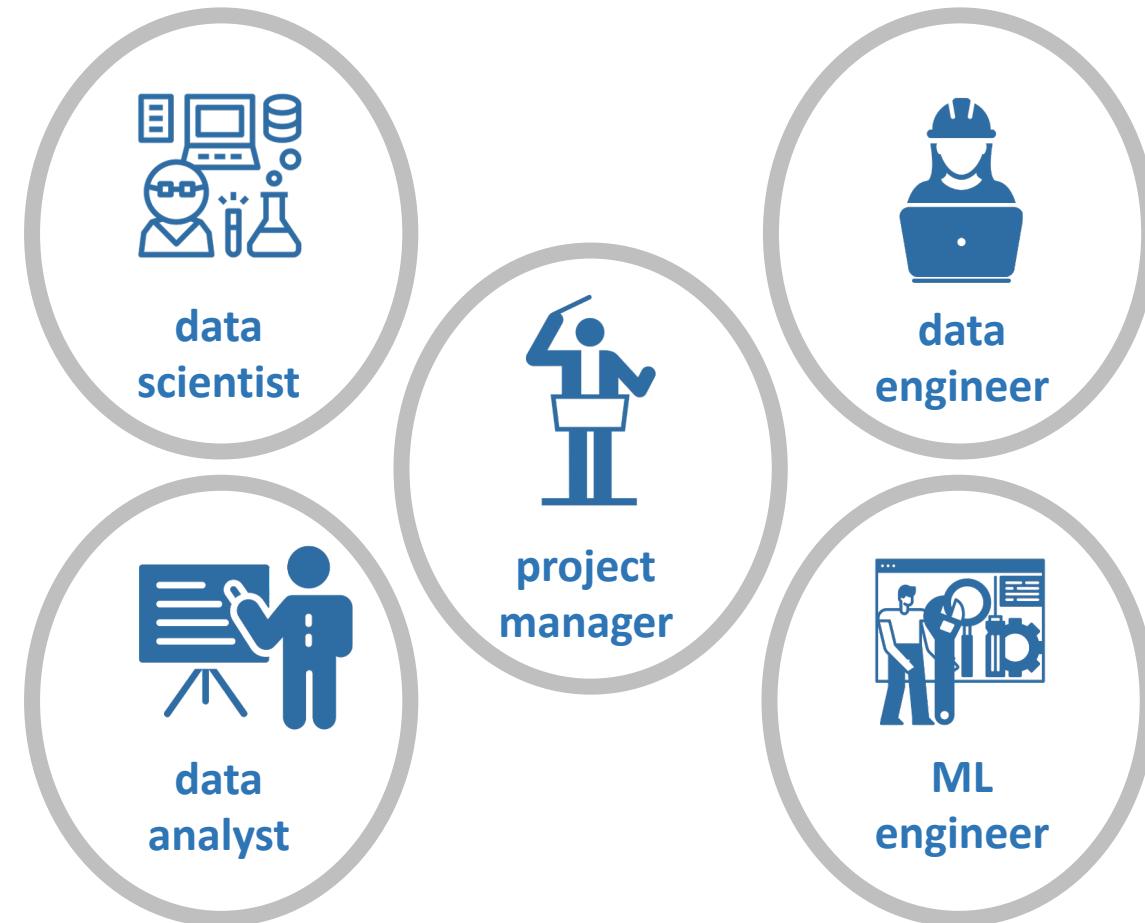
- host “**live**” solution
- deploy **redundancies**
- **end users** interact with solution

and support the Data Product lifecycle

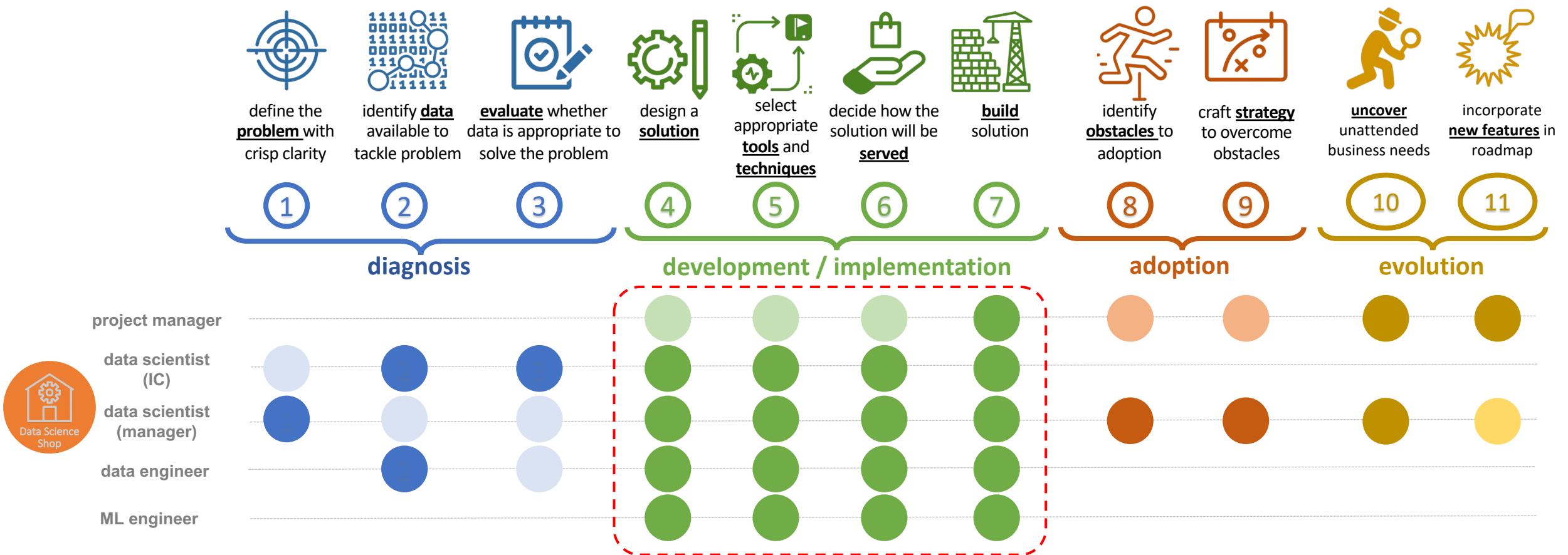


2

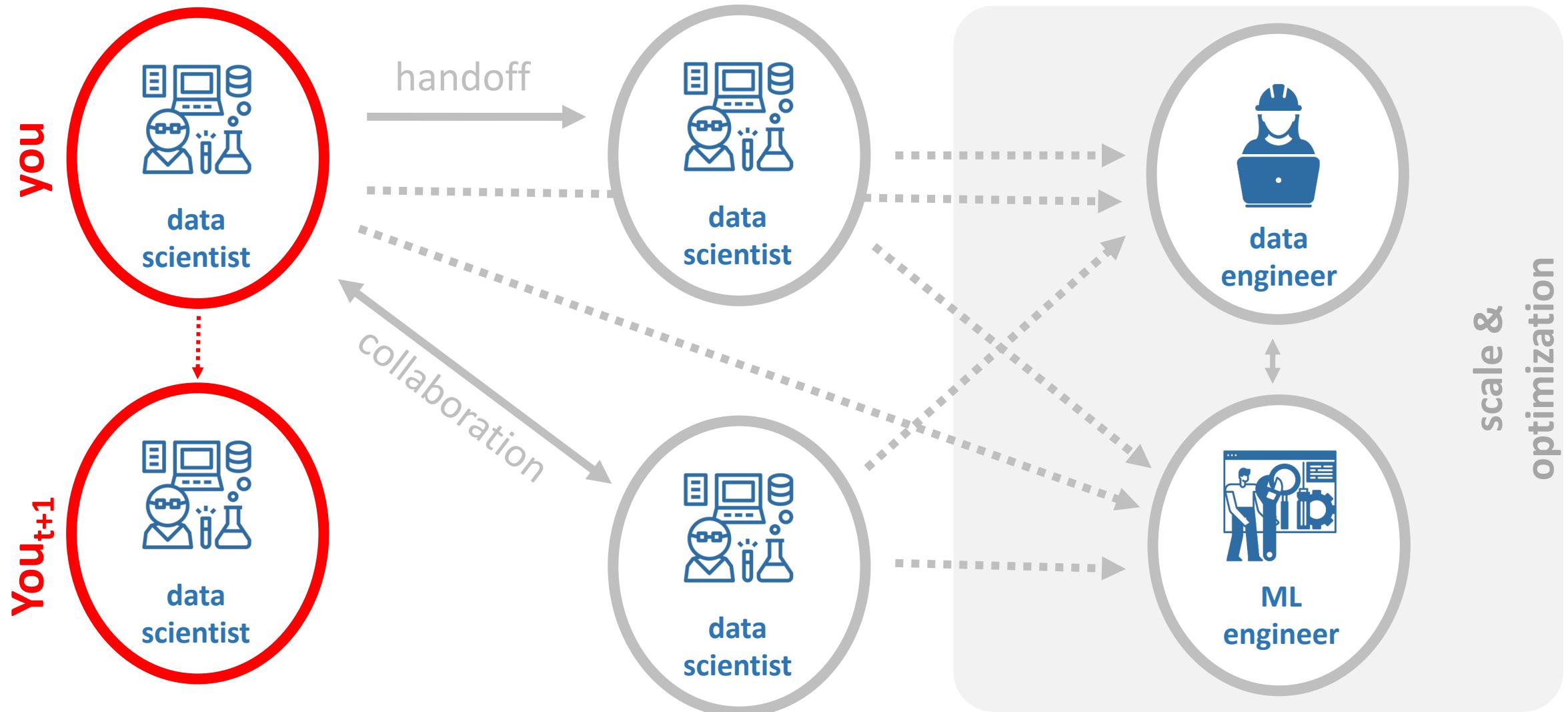
recap: collaboration to develop Data Products



recap: collaboration to develop Data Products

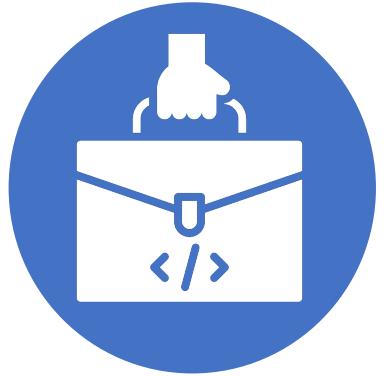


workflow collaboration in Data Science

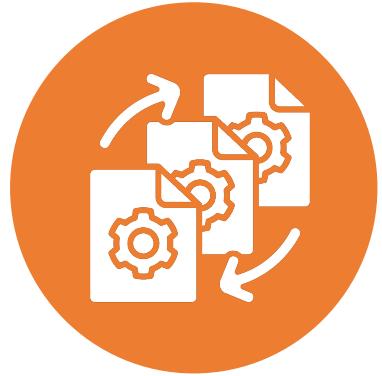


3

operational concepts in Data Science



portability



replicability



scalability

anyone should be able
to **pick up where you
left off** from any
machine

anyone should be able
to arrive at your **same
results**

your prototype should
also work for **larger
data sets** and/or be on
the path of **automation**

operational concepts in Data Science



portability



replicability



scalability

what

- flexible references
- structured and documented code
- replicate original environment

- seamless handoff
- frictionless transitions across environments

- documentation: data, software, hardware, environments
- commented code
- no manual processes

- seamless examination, review or validation
- cordial troubleshooting
- harmonious optimization

- high quality code
 - flexible functions
 - modularized code
- simplified review and validation
 - reduce time optimizing, automating and deploying

why

necessary conditions to build Data Products

1. integrate **new code** *seamlessly*
2. replace existing code *seamlessly*
3. work **simultaneously** in the same codebase *seamlessly*
4. hand off code *seamlessly*
5. travel in time (backwards and/or forwards) *seamlessly*

why do we
start here?



all Data Products involve code, and
code is the backbone of all Data Products

```
mod = modifier_obj
set mirror object to mirror
mirror_mod.mirror_object
operation == "MIRROR_X":
mirror_mod.use_x = True
mirror_mod.use_y = False
mirror_mod.use_z = False
operation == "MIRROR_Y":
mirror_mod.use_x = False
mirror_mod.use_y = True
mirror_mod.use_z = False
operation == "MIRROR_Z":
mirror_mod.use_x = False
mirror_mod.use_y = False
mirror_mod.use_z = True
selection at the end - add
ob.select= 1
ob.select=1
context.scene.objects.active = ob
mirror_obj.select = 0
bpy.context.selected_objects
data.objects[one.name].select
int("please select exactly one object")
- OPERATOR CLASSES -
types.Operator):
x mirror to the selected
```

why version control?

1. keeps “**snapshots**” of your code over time
2. expedites **code debugging** (your code and your team’s)
3. regulates **team collaboration** (everyone can see who changed what! + “air traffic control”)
4. supports the **lifecycle of a Data Product**
5. enables **reproducibility, portability, and scalability**

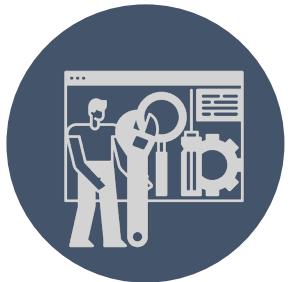
central to any activity that involves code



Software Engineering



Data Engineering



Machine Learning
Engineering

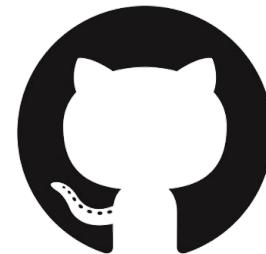


Data Science

many flavors, but we'll focus on these two



version control software



GitHub

open-source cloud service

two minutes to make sure you've:

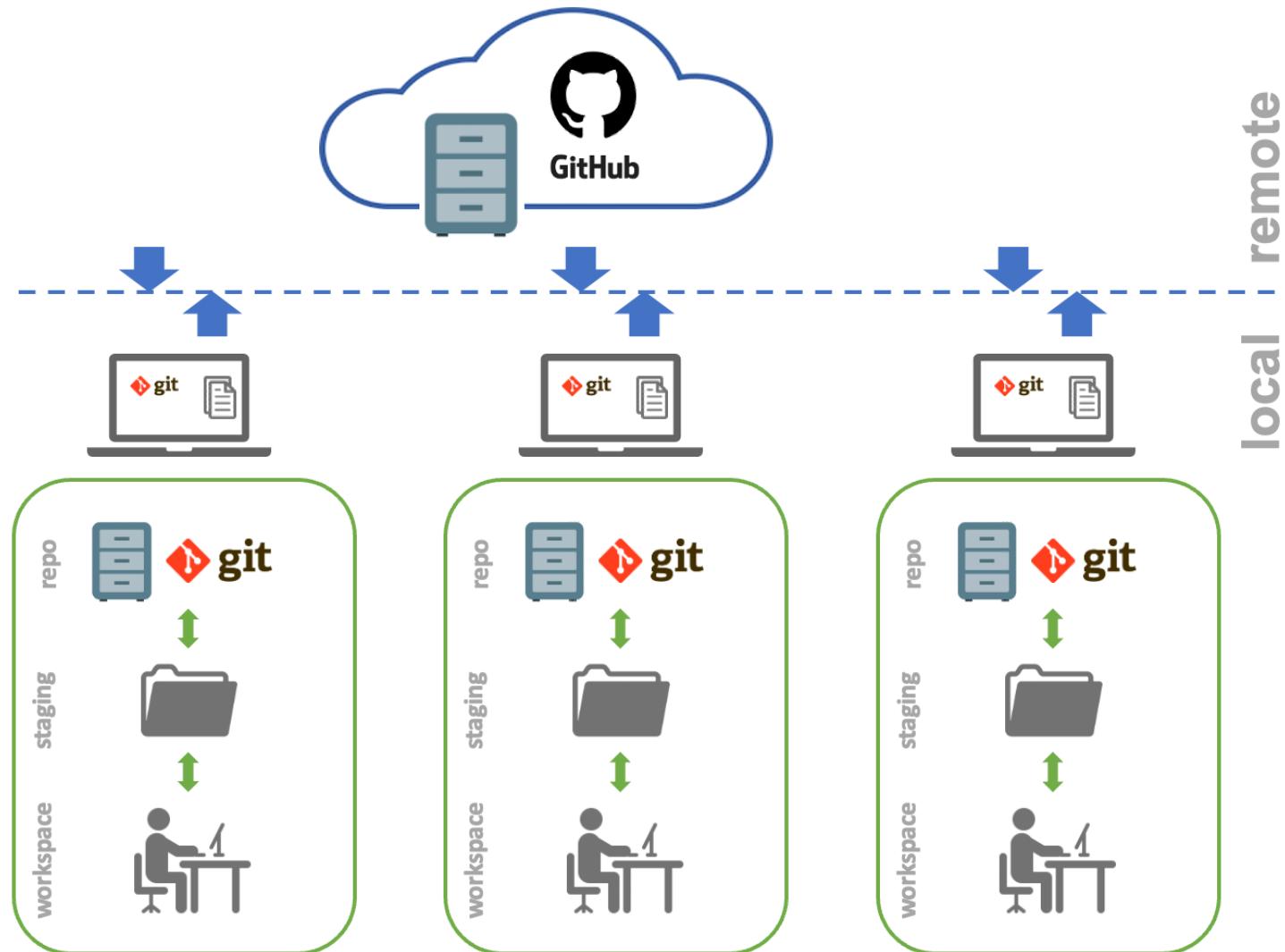
1) downloaded/installed git



2) created a GitHub account



an ideal version control setup

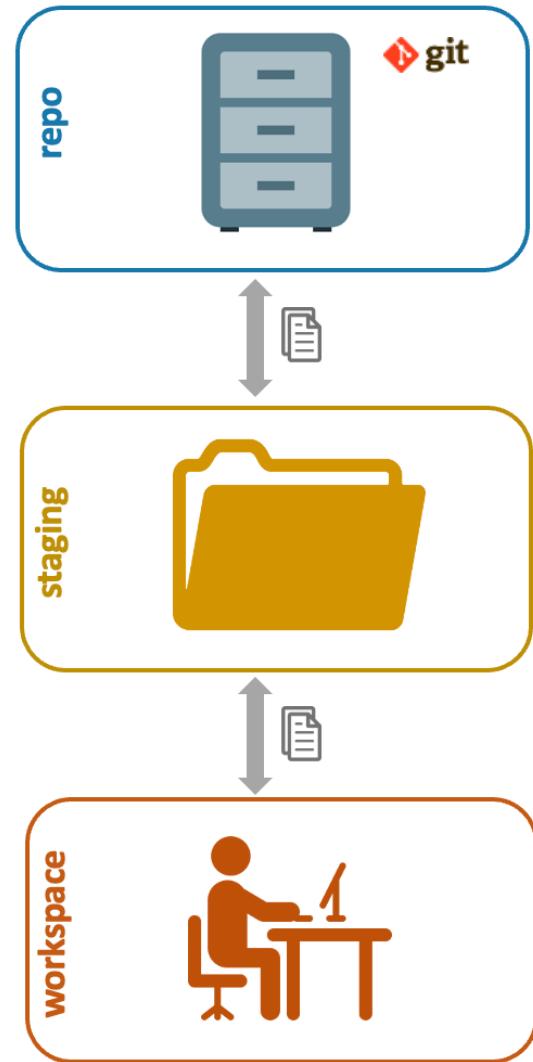


git locally

recap: what was this **git** thing?

- git is a **version control software**
 - installed “locally” on your computer (or virtual machine or computing platform)
 - keeps snapshots of your (coding) work
- helps with
 - “time travel” (insert your favorite “Back to the future” gif here)
 - keep collaboration organized when multiple people are working on the same project
- a vehicle to be nice to your fellow collaborators (and to the you of the future)

git: a mental model



git, meet your new user!

- set your **username** and **email address**

```
$ git config --global user.name "John Doe"  
$ git config --global user.email johndoe@example.com
```

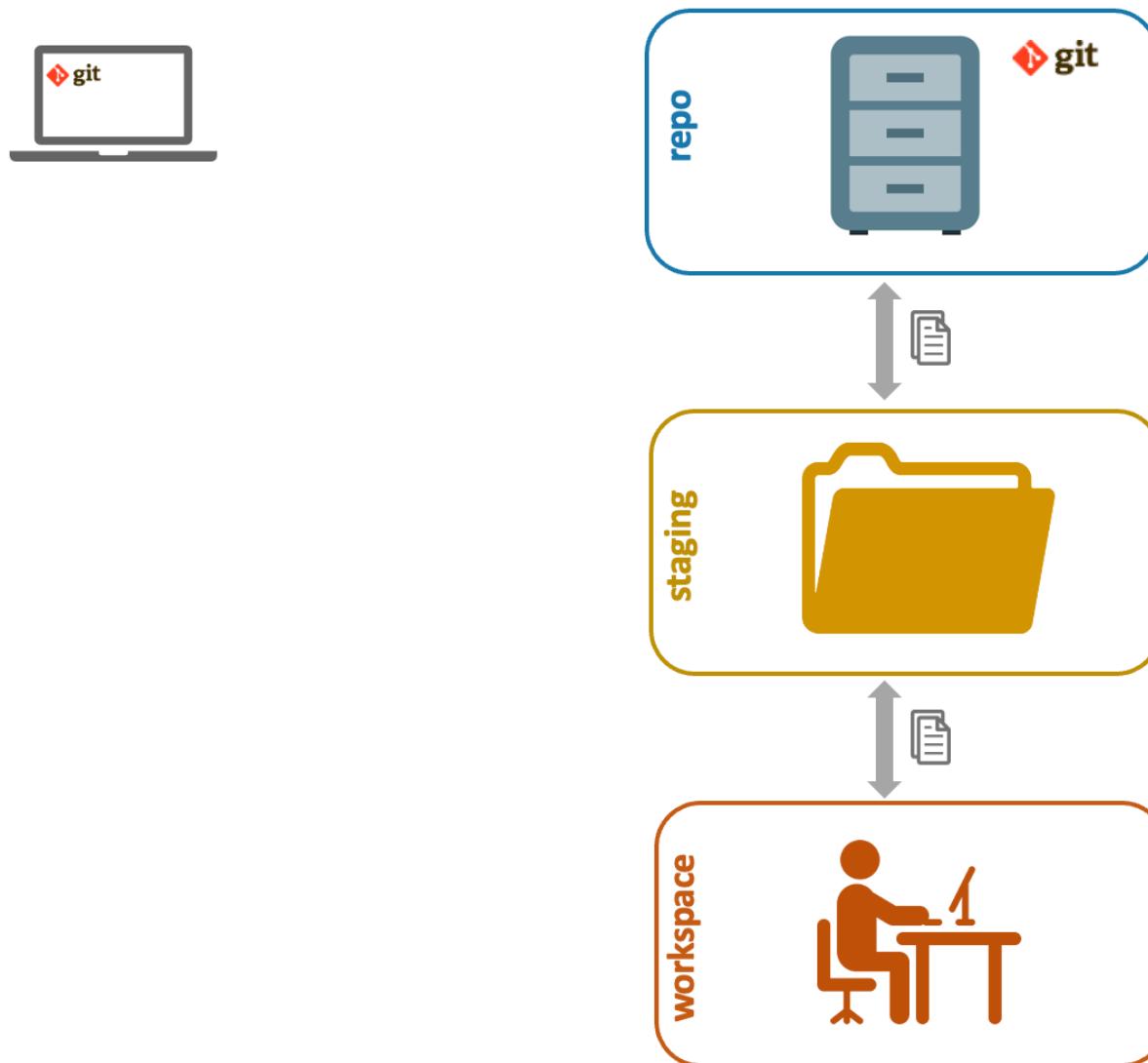
- verify that information was successfully entered

```
$ git config --list
```

- this information gets baked in your commits

ProTip: other useful information (e.g. proxy settings) also goes on `git config`

now, turn your folder structure into a git repo



now, turn your folder structure into a git repo

- go to the **root** of your project and initialize the repo

```
$ git init
```

- there are **files you never want tracked** by git (e.g. log files, access keys), even by mistake
- that's the purpose of a `.gitignore` file

now, turn your folder structure into a git repo

- from the root of your local repository, create a `.gitignore` file

```
$ touch .gitignore  
$ echo > .gitignore
```

(Mac)

(Windows)

- add files, file types and folders you want git to ignore

what do you add to a .gitignore file?

```
# OS generated files #
*.DS_Store

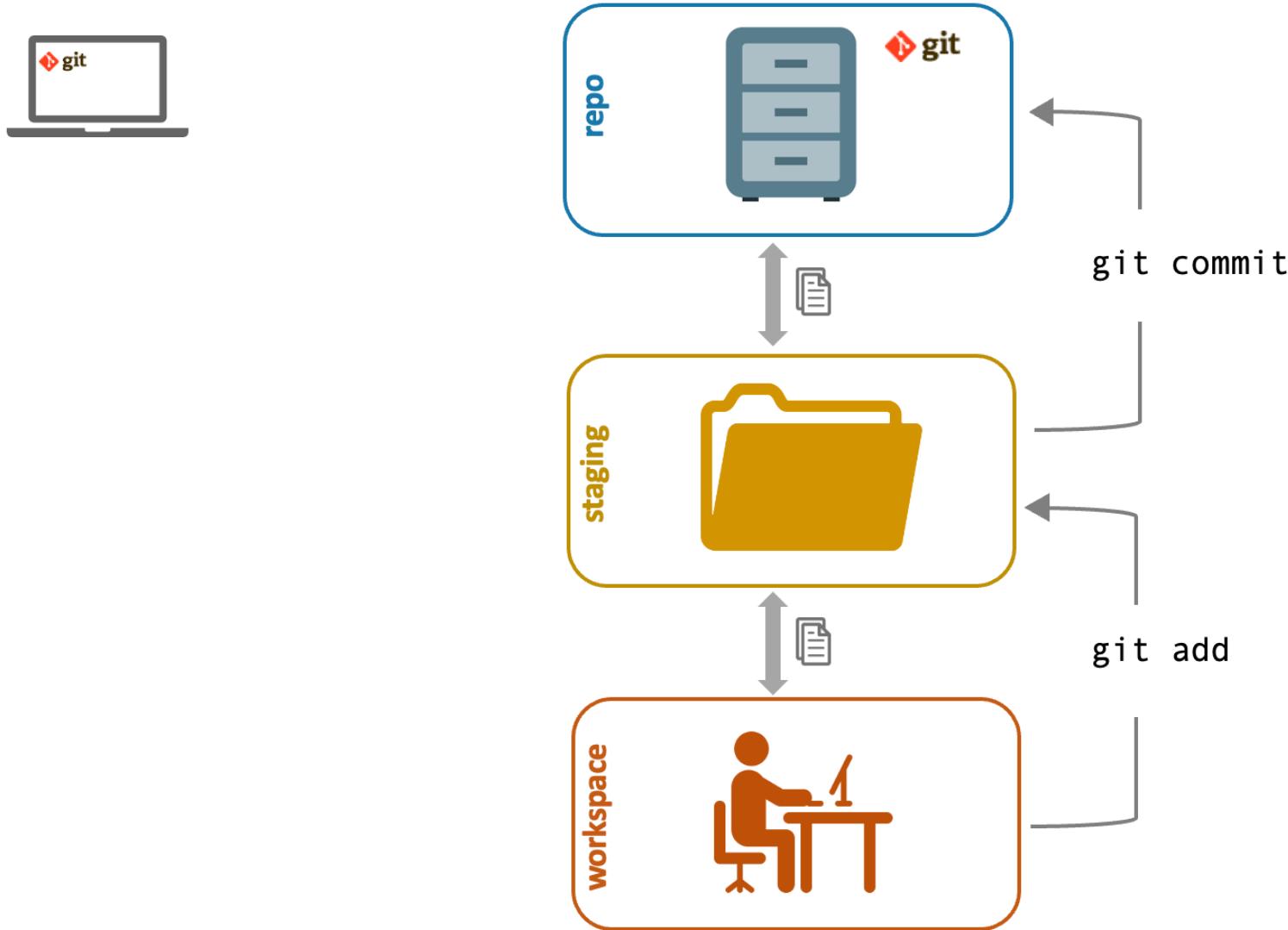
# Jupyter Notebook
.ipynb_checkpoints
```

```
# RStudio files
*.Rproj.user/

# all data folders
data/
```

ProTip: further info/templates: <https://github.com/github/gitignore>

your basic git workflow



your basic git workflow

- indicate a file to be tracked by git

```
$ git add samplefile.R
```

- verify what's being tracked

```
$ git status
```

- commit your tracked files (with an informative message)

```
$ git commit -m "Commit initial files"
```

a few confusing things about git

- a file will be committed **exactly** as it was when you git add-ed it
- if you change the file after you git add it and want to commit the new changes, you need to git add again before the git commit
- use git status to assess what's being staged and committed

git workflow ProTips

- NEVER use `git add .`
- use `git status` often as validation
- only add and commit source files
 - omit files you can reproduce using source files
- commit **small chunks of logically grouped changes**
 - you may want to undo a change, and only that change
- commit with **informative** (imperative mood) **messages**
 - *[this commit will] Rename income variable*

quick detour: what is a branch in git?



a **divergence** from the main line of development



that **isolates** development work
without affecting other branches



a branch can **merge** into another branch



repos always have a **default branch**

git workflow ProTips

- current best practice is to use `main` for your default branch; used to be `master`
- by default, git will create a `main` branch after your first commit
- easy to rename your branch to `main`

```
$ git branch -M main
```

- for a permanent solution (in git >= 2.28)

```
$ git config --global init.defaultBranch main
```

push globally
(to GitHub)

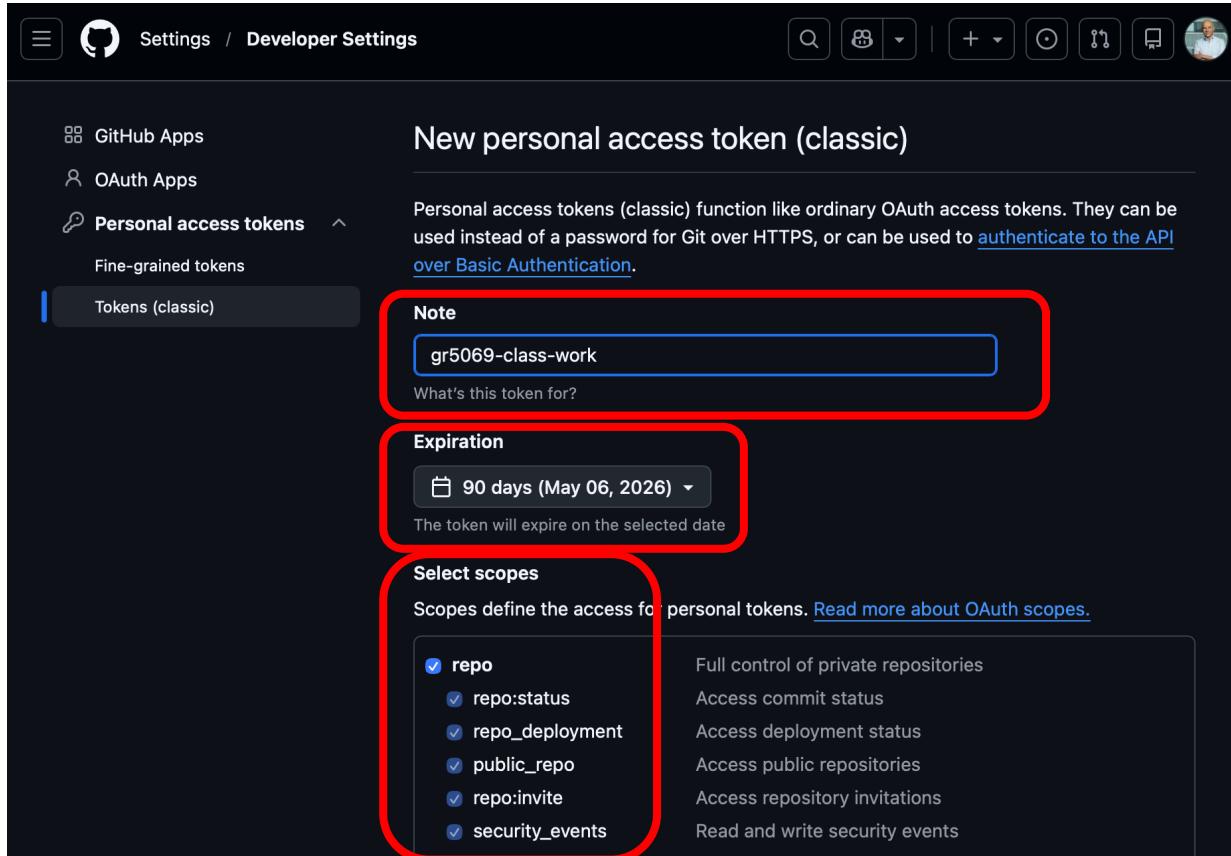
recap: what was this GitHub thing?

- **GitHub** is a **cloud service** that hosts git repositories
 - lives in the cloud
 - understands the git dialect!
 - can speak with multiple git users simultaneously
- helps with
 - persisting repository storage (your dog cannot eat your repo!)
 - synchronizing work
 - minimizing risk of people stepping on each other's toes (while working on the same project)
 - seamless transition between environments (dev > qa> staging > prod)

before we begin, you need a **personal token**

- GitHub now requires **personal tokens** with individual privileges for each user
- the token is *unique* to a specific repo
- tokens facilitate collaboration
- use your **token** as your **password** when you're working on a repo
- in the real world, a GitHub repo admin will create a token for you
- (for our purposes) you can create your token by navigating to
 - > Settings > Developer settings > Personal Access Token > Tokens (classic) > Generate new token > Generate new token (classic)

before we begin, you need a personal token



- add a meaningful name on the **Note** field
- set a delimited **Expiration** date (e.g. 90 days)
- (for our purposes) only select **repo** privileges from **Select scopes**
- **store your token!** (*you will not be able to recover it later*)

> Settings > Developer settings > Personal Access Token > Tokens (classic) >
Generate new token > Generate new token (classic)

now, create a GitHub repo

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner *  marco-morales / **Repository name *** mytestrepo  mytestrepo is available.

Great repository names are short and memorable. Need inspiration? How about [friendly-memory](#) ?

Description (optional)

 Public Anyone on the internet can see this repository. You choose who can commit.
  Private You choose who can see and commit to this repository.

Initialize this repository with:

Add a README file This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: **None** ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

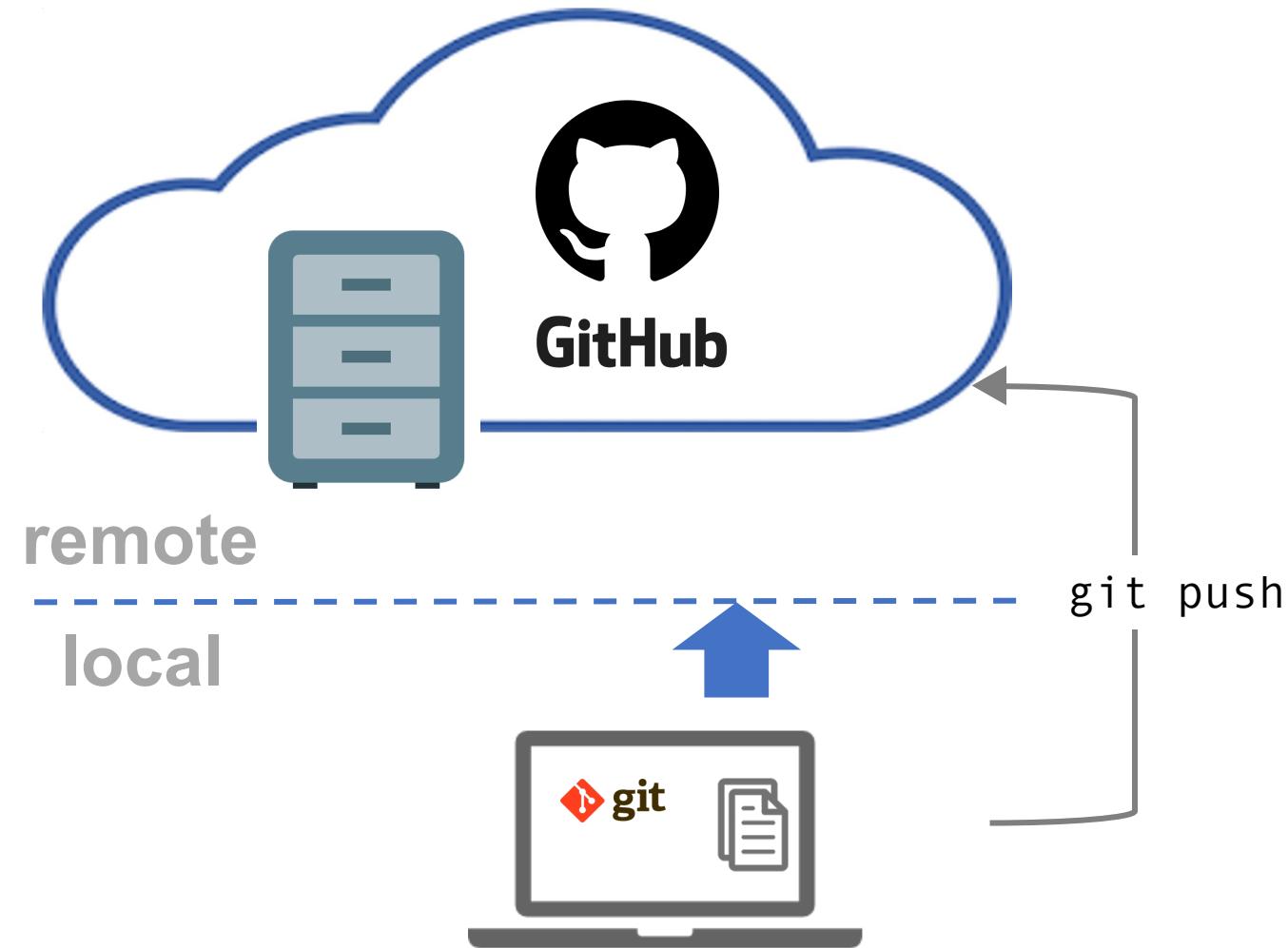
License: **None** ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

ⓘ You are creating a public repository in your personal account.

Create repository

then, push to that GitHub repo



then, push to that GitHub repo

- tell git the location of the remote GitHub repo you just created (typically nicknamed “origin”)

```
$ git remote add origin https://github.com/marco-morales/testrepo.git
```

- send committed files to your GitHub (“origin”) repo from your local git branch (“main”)

```
$ git push -u origin main
```

GitHub workflow ProTips

- current best practice is to use `main` for your default branch; used to be `master`
- by default, GitHub will create a `master` branch after you first create a repo if you do not change defaults
- easy to change permanently in your GitHub settings

