

Data Engineering Best Practices



Kerry Nakayama



Best Practices



Kerry Nakayama





Objectives

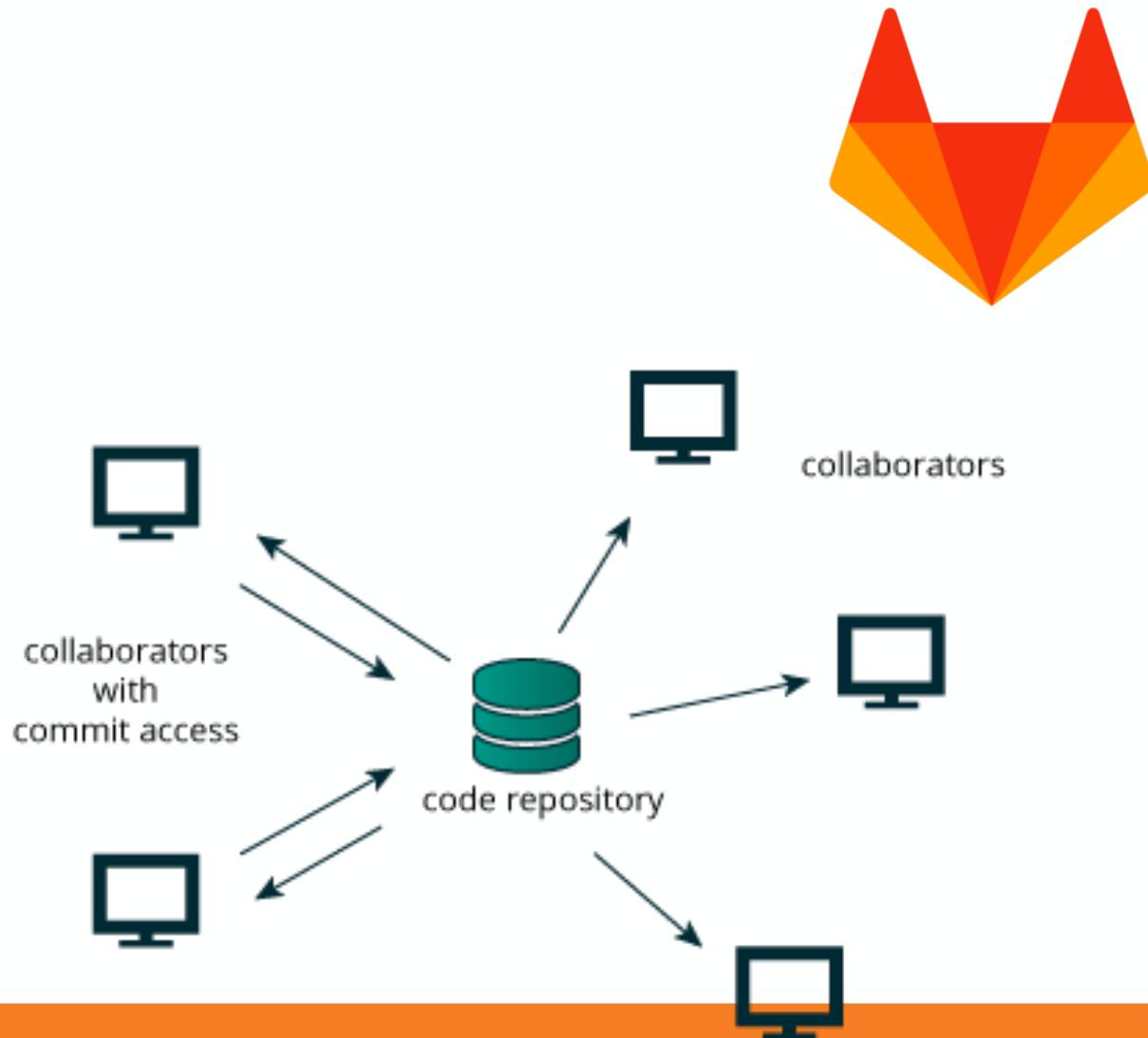
- Creating a code repository
 - Hooks
 - Linting
 - Automatic builds on push to master
 - Parallelization
- dbt integration with code repo
 - YML files
 - Testing
 - Unit testing
 - Integration testing
 - Dealing with Failed data
 - Automatic reruns





Creating Code Repository

- What are they for?
 - Version control
 - Central location for code
 - Code review – peer review
 - Data docs
 - CI/CD
 - Access control





Hooks

- What are Git Hooks?
 - Git hooks are scripts that Git executes before or after events such as: **commit**, **push**, and **receive**
 - Every Git repository has a `.git/hooks` folder with a script for each hook you can bind to
 - being able to push to your staging or production environment without ever leaving Git is just plain awesome



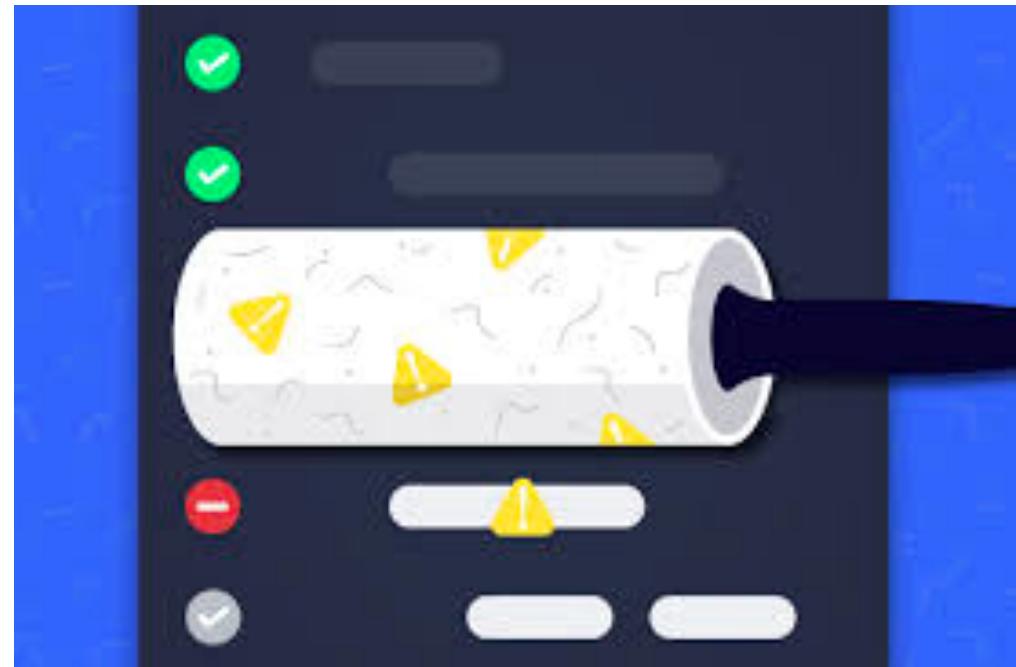


Linting



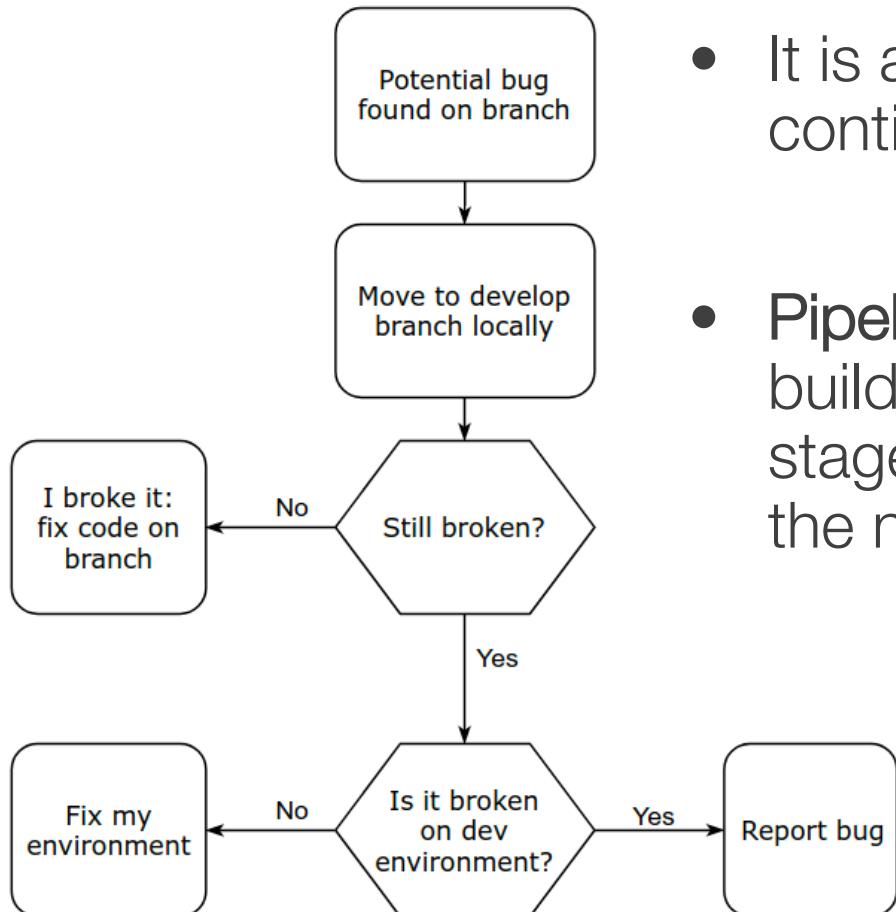
What is Code Linting?

- Best practices
 - Linting vs formatting
- Validate can be used to list all errors in the loaded data
- Use linting tool – most are open source





Automatic Builds on Push to Master



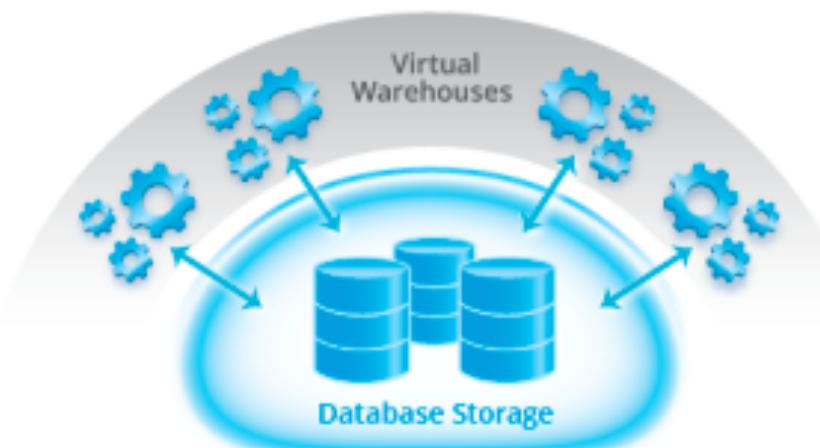
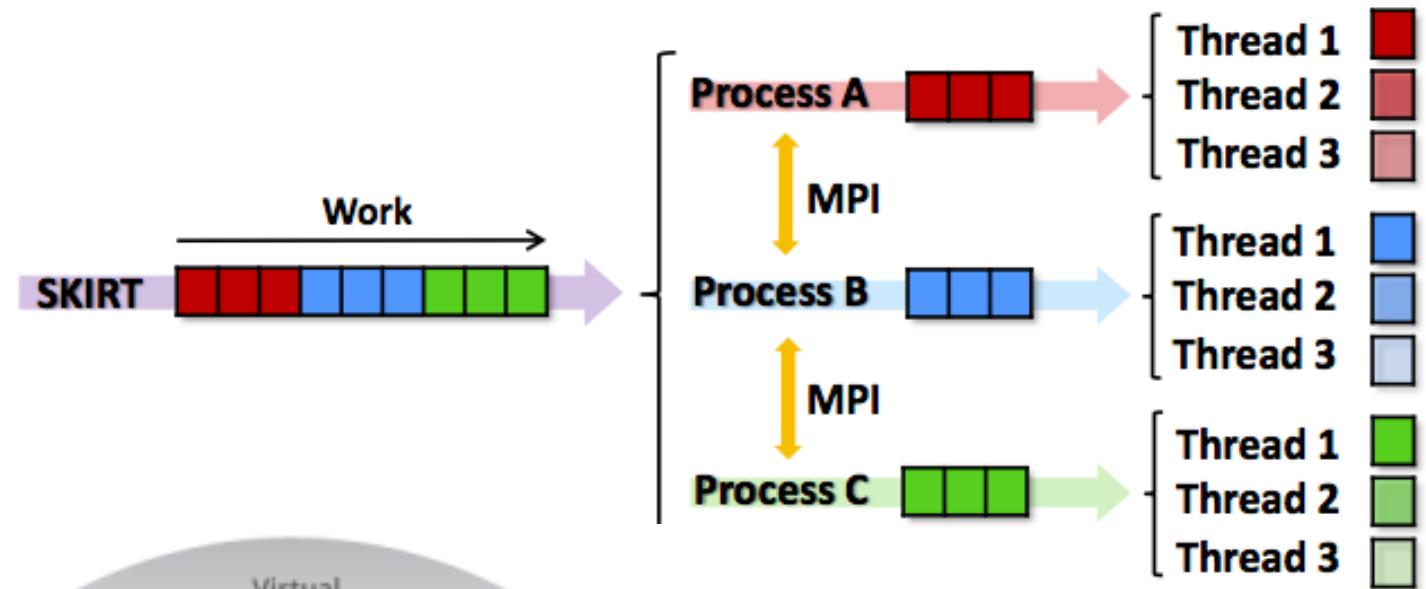
- It is arguably the most popular continuous integration tool in use today
- Pipeline - The process of automatically building code in stages – and at each stage, testing and promoting it on to the next stage





Parallelization

- What is Parallelization?
- How is it being used for your team?
- How Snowflake handles it





dbt integration with code repo



The screenshot shows the dbt Cloud interface. At the top, there's a navigation bar with a menu icon, the dbt logo, and a dropdown for 'Demo'. Below the navigation is a 'Project' section with a green 'commit...' button. Underneath, it shows the current branch: 'add-yml'. The main area displays the project structure: dbt_demo, analysis, data, dbt_modules, logs, macros, models, snapshots, target, tests, .gitignore, dbt_project.yml, and README.md.

- dbt has created a function that sets up your code repository once connected
 - dbt init – initializes folders for you to start
- Enabling Continuous Integration (CI)
 - Use Webhooks to enable runs on Pull requests

The screenshot shows the dbt Cloud Run History for Run #679064. The summary indicates a 'Success' result, triggered by 'Pull Request #72' with commit hash '#e13e39'. The run took 20 minutes, 7 seconds. The 'Details' section shows the timing of the run steps: triggered 1 week ago, in queue for 8 seconds, ran for 19 minutes, 59 seconds, and finished 1 week ago. The 'Run Steps' section lists three steps: 'Clone Github Repository' (SUCCESS - 00:00:00), 'Create Profile from Connection Garage - Redshift (override schema to 'sinter_pr_101_72')' (highlighted with a red box and SUCCESS - 00:00:00), and 'Invoke dbt with `dbt deps`' (SHOW LOGS +).



YML Files

- dbt_project.yml
 - Every [dbt project](#) needs a dbt_project.yml file
 - this is how dbt knows a directory is a dbt project
 - It tells dbt how to operate on your project
 - When dbt runs, it will search through the source-paths directories of your dbt project looking for any files that end with .yml

```
dbt_project.yml

name: string
config-version: 2
version: version
profile: profilename
source-paths: [directorypath]
data-paths: [directorypath]
test-paths: [directorypath]
analysis-paths: [directorypath]
macro-paths: [directorypath]
snapshot-paths: [directorypath]
docs-paths: [directorypath]
target-path: directorypath
log-path: directorypath
modules-path: directorypath
clean-targets: [directorypath]
query-comment: string
require-dbts-version: version-range | [version-range]
quoting:
  database: true | false
  schema: true | false
  identifier: true | false
models:
  <model-configs>
seeds:
  <seed-configs>
snapshots:
  <snapshot-configs>
sources:
  <source-configs>
on-run-start: sql-statement | [sql-statement]
on-run-end: sql-statement | [sql-statement]
```



Testing

- dbt makes it easy to define automated tests on your models

Tests are gifts you send to your future self

Testing is an essential part of a [mature analytics workflow](#). It only takes a minute to set up a test -- your future self will thank you!

- Schema Tests

- Schema tests are assertions that a model's schema adheres to basic rules: referential integrity, uniqueness, and nullity, for instance

- Custom data tests

- Data tests are sql SELECT statements that return 0 rows on success, or more than 0 rows on failure

```
schema.yml          schema.yml          schema.yml          schema.yml
version: 2          version: 2          version: 2          version: 2
models:             models:             models:             models:
- name: people      - name: account_id   - name: status     - name: people
  columns:           tests:             tests:             columns:
  - name: id         - not_null        - not_null       - name: id
  tests:             - unique          - not_null       tests:
  - not_null         - relationships:  - unique          - to: ref('accounts')
                                         - field: id
```



Examples of Errors

Failed Value Conversions
37%

Data Warehouse
Errors
19%

Missing Required
Values
11%

Changed Schemas
11%

Custom Logic
Failures
9%

Strings to
Integers
9%

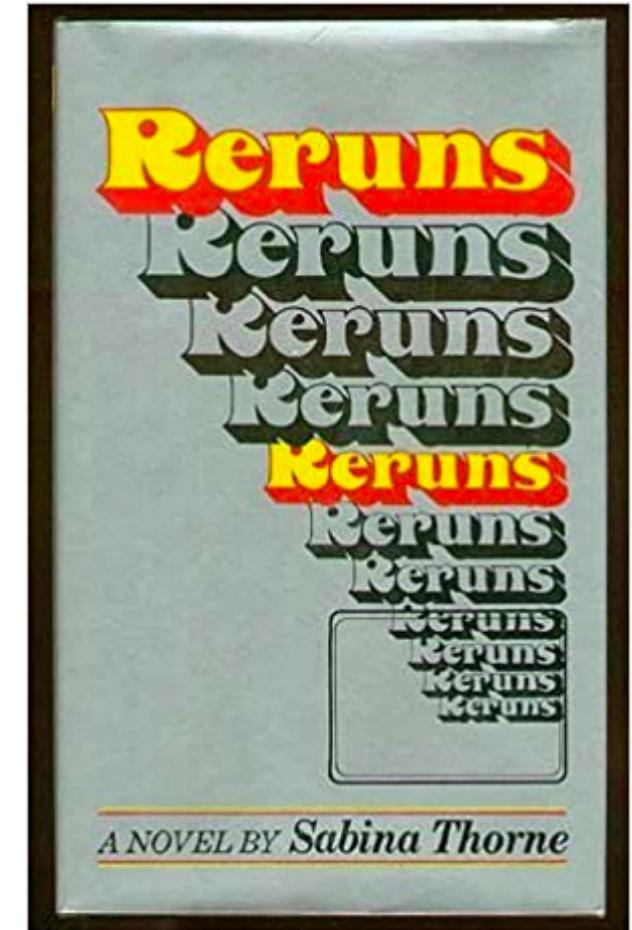
Overly long
strings
5%



Automatic Reruns



- 3rd Party failures for pulling the data happen all the time
- Tests in dbt to check for a data failure
 - Failures include
 - No data
 - Duplicate data – test for uniqueness
 - Incomplete data – expecting x rows – warning
- Set up a trigger to kick off a scripting loop
 - Python or jinja code can be used to create loops





Confused? ASK QUESTIONS





Break & Lab 3

