Github Repository as Project Deliverable

MSEN 489/655 - Materials Design Studio

Juan E Flórez-Coronel

Department of Materials Science and Engineering
Texas A&M University
College Station, TX

Spring 2026

Outline

Introduction

Why GitHub for Materials Design Studio?

- Your repository **is the deliverable**: code, data handling, results, and the story of your work.
- Git tracks **what changed, when, and why** (useful for teamwork and grading).
- GitHub makes collaboration explicit: **branches, pull requests, reviews, issues**.
- Reproducibility matters: someone should be able to **clone, create the conda env, and rerun key results**.

Learning Objectives

- Navigate and evaluate an existing repo quickly (what it does, how to run it).
- Use a safe workflow: **branch** → **commit** → **PR** → **merge**.
- Build a **complete course deliverable repo** for Python/conda projects.
- Write a strong `README.md` and a useful `AGENTS.md` (AI-agent playbook for using the repo).

Git vs GitHub (quick mental model)

- **Git** = version control tool (snapshots, branches, merging, history).
- **GitHub** = hosting + collaboration (PRs, issues, reviews, releases).
- Repository = project folder + time machine + collaboration record.

What graders (and colleagues) want

- **Clear structure**: where code lives, where figures live, where the report lives.
- **Reproducible environment**: `environment.yml` (conda) + instructions.
- **Reproducible results**: one or two commands to regenerate key plots/tables.
- **Documentation**: `README.md` tells the story; `AGENTS.md` tells agents how to operate the repo safely.
- **Professional workflow**: branches + PRs; minimal chaos on `main`.

Repositories

Repo Tour

- **README.md**: What is this? How do I install/run? How to reproduce results?
- **Environment**: `environment.yml`, `requirements.txt`, `pyproject.toml`
- **Structure**: `src/`, `notebooks/`, `data/`, `results/`, `report/`
- **Signals of quality**: tests, CI badges, issues/PR activity, releases/tags
- **License**: what you are allowed to reuse (don't copy blindly)
- **AGENTS.md (if present)**: what an automated agent should run, where outputs go, and guardrails

Clone and Inspect

- Install Git.

```
git clone github.com/rarroyave/MSEN655-Materials_Design_Studio.git
cd MSEN655-Materials_Design_Studio

# Your dashboard:
git status

# Explore:
ls
cat README.md
```

- If you only remember one command: `git status`.
- Goal: in 2 minutes, you can explain **what it does** and **how to run it**.

Github Account Authentication

- To create a new repo:
  - Create on Github Online
  - Install Github CLI (gh)
  - Use PAT or SSH Key
- To use an existing repo:
  - Use IDE's Built-In Github Authentication
  - Use PAT or SSH key
- You will also need to configure in Git:

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

Lets Create a Repo

- If you already installed and authenticated your account in Git and Github CLI (gh):

```
mkdir exampleRepo
cd exampleRepo
vim README.md
git init
git add .
git commit -m "Initial commit"
gh repo create
```

Git Workflow

The safe loop: branch → commit → PR → merge

- Keep `main` stable and runnable.
- Do work in feature branches (one branch per task).
- Open a pull request (PR) to merge into `main`.
- Review quickly (even within the team), then merge.

Essential commands

```
# Inspect
git status
git diff
git log --oneline

# Branching
git checkout -b feature/my-task
git checkout main
git merge feature/my-task
```

```
# Save a snapshot
git add <file_or_folder>
git commit -m "Meaningful message"

# Sync with GitHub
git push
git pull
```

Commit messages engineers should write

- Bad: `"updates"`, `"fix"`, `"final"`, `"stuff"`
- Good: `"Add DOE sampler for Latin hypercube baseline"`
- Good: `"Refactor feature extraction into src/features.py"`
- Good: `"Reproduce Figure 2 with new preprocessing pipeline"`
- Rule: message should answer **what** changed and ideally **why**.

Branches: what to name them

- Use descriptive names:
  - `feature/doe-screening`
  - `feature/bo-loop`
  - `fix/data-loader`
  - `docs/readme-install`
- Avoid: `test`, `new`, `branch1`, `juanes-branch`

Merging and conflicts

- A merge conflict means: Git needs a human decision (normal in teams).
- Typical cause: two people edit the same lines in the same file.
- Resolve conflicts by editing the file to the final desired content, then:

```
git add <resolved_file>
git commit -m "Resolve merge conflict in <file>"
```

PRs for class projects

- A PR is a proposal: "merge my branch into main."
- PRs create a record of:
  - who contributed what,
  - discussion and decisions,
  - review/approval,
  - the exact diff.
- Even small teams benefit: PRs reduce breakage and last-minute chaos.

Minimal PR template

```
Title: <short verb phrase>
What changed?
- ...
Why?
- ...
How to test / reproduce?
- conda env create -f environment.yml
- conda activate <env>
- python -m src.run_experiment --config configs/base.yaml
Notes:
- Any caveats, runtime, data location, etc.
```

Deliverable Repo

A complete deliverable repo

- **README.md**: story + setup + run + reproduce steps
- **AGENTS.md**: instructions for AI agents to navigate, run, and modify the repo safely
- **environment.yml**: conda environment for reproducibility
- **Source code**: organized (notebooks are okay, but core logic should live in `src/`)
- **Report/Slides**: final PDF(s) in `report/`
- **Data policy**: small sample data in repo; full data referenced externally (if needed)
- **Results**: either generated on demand, or stored carefully (avoid huge binaries)

Recommended repo structure (Python/conda)

```
materials-studio-project/
  README.md
  AGENTS.md
  environment.yml
  .gitignore
  src/
    __init__.py
    data/
    models/
    bo/
    plots/
    run_experiment.py
  notebooks/
    01_exploration.ipynb
    02_results_figures.ipynb
  configs/
    base.yaml
  report/
    final_report.pdf
    final_slides.pdf
  data/
    sample/          # small sample only
  results/
    figures/         # optionally generated
```

Data and results: what belongs in Git?

- **Yes**: small sample data for demos/tests; processed tables (small); configs; scripts.
- **Maybe**: generated figures (if small and required for the report).
- **No**: raw large datasets, multi-GB files, model checkpoints unless tiny.
- Best practice:
  - keep heavy data external (Drive/HPC storage),
  - document how to obtain it and where to place it,
  - include a `data/README.md` explaining the policy.

Never commit secrets

- No API keys, passwords, tokens, private URLs with credentials.
- Use environment variables and keep `.env` out of Git.
- Put sensitive patterns in `.gitignore`.

Why conda in this studio?

- Materials workflows often depend on compiled libraries (numpy/scipy), ML stacks, and system deps.
- Conda environments reduce "it works on my machine" problems.
- Your goal: graders can reproduce key results on a clean machine by following README steps.
- "From scratch" instructions in README.md should work:
  - clone repo
  - create conda env
  - run one command to reproduce key plots/tables
- If results take a long time:
  - provide a "quick run" mode (small sample / fewer iterations),
  - clearly separate "full run" vs "demo run."

environment.yml recommended minimal example

```
name: msen655-project
channels:
  - conda-forge
dependencies:
  - python=3.11
  - numpy
  - scipy
  - pandas
  - matplotlib
  - scikit-learn
  - pip
  - pip:
      - -e .
```

- Use `conda-forge` consistently if possible.
- If you package your project, `-e .` installs it editable.

Conda commands students should know

```
# Create environment from file
conda env create -f environment.yml

# Activate
conda activate msen655-project

# Update environment after editing environment.yml
conda env update -f environment.yml --prune

# Export (if needed)
conda env export --no-builds > environment.yml
```

README.md: what it must accomplish

- Explain the project in plain language (what problem, what approach).
- Provide install/setup steps (conda).
- Provide run steps (commands).
- Provide reproduction steps for the **main results** (figures/tables).
- Document repo structure and where to find the final report.
- Name the team and contributions (briefly).

README.md recommended sections

- Project Title + One-sentence summary
- Motivation / problem statement (2–5 lines)
- Method overview (bullets)
- Repository structure (map)
- Setup (conda)
- How to run (commands)
- Reproduce key results (exact steps)
- Data access policy (where data lives)
- Team and contributions
- Citation / acknowledgements (if you built on external code)

README tips for engineering projects

- Prefer **commands** over paragraphs: graders should copy/paste and run.
- Include runtime notes: "quick run $\sim$ 2 minutes", "full run $\sim$ 2 hours".
- If notebooks are required:
  - keep a "results" notebook that cleanly runs top-to-bottom,
  - avoid hidden state; restart kernel and run all before submission.
- Link directly to `report/final_report.pdf` and any key figures.

AGENTS.md: what it is and why it matters

- `AGENTS.md` is a **machine-readable, human-friendly playbook** for AI assistants/agents.
- Purpose: enable an agent to **use the repository correctly**:
  - understand the repo map and entry points,
  - set up the conda environment,
  - run experiments and regenerate results,
  - make safe edits following project conventions.
- Think: "If an agent joined our team today, could it run the pipeline without guessing?"

What to include in AGENTS.md

- **Quickstart commands** (clone, conda, one command to run)
- **Repo map**: where core logic lives, where configs live, where outputs go
- **Entry points**: canonical scripts/modules to run (avoid running random notebooks)
- **Config system**: how to change experiment settings without editing code
- **Data policy**: where large data is expected and what is tracked vs not tracked
- **Guardrails**: what the agent should *not* do (secrets, huge files, force pushes)
- **Definition of done**: how to verify success (expected files, tests, checks)

AGENTS.md best practices

- Keep it **actionable**: commands, paths, expected outputs.
- Keep it **authoritative**: one canonical way to run the pipeline.
- Update it whenever:
  - repo structure changes,
  - entry points change,
  - environment changes,
  - output locations change.
- Align it with `README.md`: humans and agents should follow the same workflow.

"Runnable" Repo

Prefer scripts for pipelines; notebooks for exploration

- Notebooks are great for exploration and figure polishing.
- But deliverables should be runnable via:
  - **scripts** in `src/` and
  - **configs** in `configs/`.
- Goal: a single command can run an experiment consistently.

A simple "entry point" pattern

```
python -m src.run_experiment --config configs/base.yaml
python -m src.plots.make_all_figures --config configs/base.yaml
```

- Use a config file to avoid editing code for each run.
- Keep outputs in `results/` (and document it in README and AGENTS).

What to include in configs (example)

- Data path(s) and preprocessing options
- DOE or BO settings (budget, seed, acquisition function)
- Model hyperparameters (if fixed) and evaluation metrics
- Output folder and naming conventions
- Random seeds (for repeatability)

Final Submission

Submission strategy

- Freeze a graded version using a **tag or GitHub Release**:
  - Tag name: `final` or `v1.0`
  - Ensures the repo state is fixed at submission time.
- Ensure instructors/TAs have access:
  - public repo OR
  - private repo with collaborators added.

Final checklist

- `README.md` contains setup + run + reproduce steps (copy/paste runnable).
- `AGENTS.md` contains the authoritative repo map, entry points, guardrails, and verification steps.
- `environment.yml` creates successfully on a clean machine.
- `report/final_report.pdf` exists and matches the repo results.
- `main` is clean: no broken code, no debug junk, no huge files.
- Data policy is explicit (sample data included; full data instructions provided).

A "clean submission" workflow

```
# 1) Sync main
git checkout main
git pull

# 2) Run your reproduction steps locally
conda env create -f environment.yml
conda activate msen655-project
python -m src.run_experiment --config configs/base.yaml --quick
python -m src.plots.make_all_figures --config configs/base.yaml
```

A "clean submission" workflow

```
# 3) Commit any final doc fixes
git add README.md AGENTS.md report/final_report.pdf
git commit -m "Finalize submission docs and report"
git push

# 4) Tag the final version (optional but recommended)
git tag -a final -m "Final submission for MSEN 655"
git push origin final
```

What to avoid in the last 24 hours

- Merging huge PRs without testing locally.
- Changing environment dependencies without verifying a clean install.
- Committing large datasets or binaries (may break cloning or exceed limits).
- Editing figures manually without documenting how they were generated.
- Letting README/AGENTS drift from reality (agents/humans should not have to guess).

Troubleshooting and Recovery

Common problems and fast fixes

- "My changes aren't showing up" → `git status`, then `git add`, `git commit`, `git push`
- "Push rejected" → `git pull` (resolve conflicts if needed) then `git push`
- "I want to discard local edits" → `git restore <file>`
- "We broke main" → revert or fix via a PR, do not panic-commit on main

If your repo is messy: stabilization plan

- Freeze `main`: no direct commits; PRs only.
- Create a `release/final-cleanup` branch.
- Focus on:
  - README accuracy,
  - AGENTS accuracy,
  - environment reproducibility,
  - clean run scripts,
  - report PDF correctness.
- Merge cleanup PR and then tag `final`.

Conclusion

Summary

- A strong deliverable repo is **runnable, reproducible, and readable**.
- `README.md` is your user manual; `AGENTS.md` is your AI-agent operating manual.
- Use a safe workflow: **branches + PRs** keep `main` stable.
- Conda environment + clear run commands turn your repo into a professional deliverable.

Thank You!

Questions?