ICE 1: Founding the Ministry of Jokes

ICE 1: MoJ Kickoff - Repo & Round-Trip Sync

- **Objective:** To create the team's "Ministry of Jokes" (MoJ) repository, test all members' Git synchronization, deploy and verify the test scaffolding, and sync the minimal application code.
- **Time Limit:** 25 minutes
- **Context:** This is a high-fidelity simulation of a "Day 0" professional workflow. We will sequentially de-risk our project: first, testing our connections (Sync 1), then verifying our test harness (the ICE 2 dependency), and finally, syncing our first "feature" (the app).

Role Kit Selection (Strategy 1: Parallel Processing ⚡)

For this ICE, we will use the **Founding Engineer's Kit**. Assign these three roles immediately. *Remember the course policy: You cannot hold the same role for more than two consecutive weeks.*

- **Repo Admin:** (GitHub) Creates the repository on `github.iu.edu` and manages collaborators.
- **Process Lead:** (Git & Scaffolding) Creates the feature branch, pushes the *initial scaffolding* (test harness, requirements), and later pushes the final `CONTRIBUTIONS.md` log.
- **Dev Crew:** (Code & Verification) Verifies the scaffolding, *then* creates and verifies the `app.py` feature.

Task Description: Project Ignition

Part 1: Repository Creation

- **([Repo Admin])**
  1. Navigate to **`github.iu.edu`** (our private, enterprise GitHub).
  2. Create a new **private** repository.
  3. **Repository Name:** `FA25-SE1-TeamXX-moj` (where `XX` is your two-digit team number).
  4. **Initialize this repository with a `README.md` file.**
  5. **Select the MIT License. This will create a `LICENSE` file in your repo.**
  6. Go to `Settings` $\rightarrow$ `Collaborators`.
  7. Add every member of your team, your **Team TA**, and the **Instructor (`@seiffert`)** as collaborators.

Part 2: Initial Clone

- **([All Team Members])**
  1. `git clone` the new repository using SSH: `git clone git@github.iu.edu:FA25-SE1/FA25-SE1-TeamXX-moj.git`
  2. `cd` into the new `FA25-SE1-TeamXX-moj` directory.
  3. You should all see the `README.md` file on the `main` branch.

Part 3: Branch & Scaffolding Push

- **([Process Lead])**
  1. Create the feature branch: `git checkout -b ice1-repo-setup`
  2. **Create the scaffolding files:** Copy/paste the contents below into the correct new files. **NOTE:** Check your indentation when pasting from the ICE PDF.
     - Create `.gitignore`

```
# Python
venv/
*.pyc
__pycache__/
.env

# IDE
.vscode/
.idea/
```

     - Create `requirements.txt` (This is a critical dependency for ICE 2)

```
flask
pytest
```

     - Create `tests/` directory: `mkdir tests`
     - Create `tests/test_app.py` (This is a critical dependency for ICE 2)

```
def test_placeholder():
    """
    This is a placeholder test to ensure pytest
    passes on the initial CI run in ICE 2.
    """
    assert True
```

  3. Commit and push this scaffolding:

```
git add .
git commit -m "chore: add branch, gitignore, and test scaffold"
git push -u origin ice1-repo-setup
```

  4. **Announce to your team:** "**Sync 1!** Pull the new branch and scaffolding!"

Part 4: Synchronization Test 1 (Scaffolding Pull)

- **([All Team Members])**
  1. Fetch all new branches: `git fetch`
  2. Check out the feature branch: `git checkout ice1-repo-setup`

3. Pull the new files: `git pull`
4. **Goal:** You must see `.gitignore`, `requirements.txt`, and `tests/test_app.py` appear locally. If this fails, **stop and ask a TA for help.** This step clears both the SSH blocker and the ICE 2 dependency.

Part 5: App Creation & Local Verification

- **([Dev Crew])**
  1. **Verify the Scaffolding:**
     1. Create the virtual environment: `python3 -m venv venv`
     2. Activate it: `source venv/bin/activate` (You should see `(venv)` in your prompt).
     3. Install dependencies: `pip install -r requirements.txt`
     4. **Verify Tests (ICE 2 Dep):** Run `pytest`. You **must** see one test pass. This proves the *scaffolding* is functional.
  2. **Create the Application Code:**
     - **Pedagogical Maxim:** "Verify scaffolding, *then* add features." Now that you've proven the test harness works, create the `app.py` file. (Feel free to give your MoJ its own persona.)

```
from flask import Flask, jsonify

app = Flask(__name__)

@app.route("/")
def hello_world():
    """
    A simple 'Hello World' endpoint to verify the app is
running.
    """
    return jsonify({
        "message": "Ministry of Jokes is now open!",
        "version": 0.1
    })
```

  3. **Verify the Application Code:**
     - **Pedagogical Maxim:** "Verify locally, *then* push."
     - Run `flask --app app run --debug`. You **must** see the server running.
  4. Once both `pytest` and `flask run` are verified, commit and push the app:

```
git add app.py
git commit -m "feat: add initial flask app"
git push
```

  5. **Announce to the Process Lead:** "App is pushed and verified!"

Part 6: Process Logging (Sync 2)

- **([Process Lead])**
    1. **Wait for the Dev Crew's announcement.**
    2. Pull their new file: `git pull` (You should see `app.py` appear).
    3. Create a file named `CONTRIBUTIONS.md`.
    4. Add the log entry (see template below) to this file.
    5. Commit and push this file:

    ```
    git add CONTRIBUTIONS.md
    git commit -m "docs: log ICE 1 roles and work"
    git push
    ```

    6. **Announce to your team:** "**Sync 2!** Pull the final log file!"

Part 7: Final Team Synchronization

- **([All Team Members])**
    1. Run `git pull`.
    2. **Goal:** Everyone on the team must have the final `CONTRIBUTIONS.md` file on their local machine. When you run `git pull` and see "Already up to date," you are synchronized.

`CONTRIBUTIONS.md` Log Entry

One team member (Process Lead) share their screen. Open `CONTRIBUTIONS.md` on your feature branch (`ice1-repo-setup`) and add the following entry **using this exact format**:

```
#### ICE 1: MoJ Kickoff - Repo & Round-Trip Sync
* **Date:** 2025-10-20
* **Team Members Present:** `@github-user1`, `@github-user2`, ...
* **Roles:**
    * Repo Admin: `@github-userX`
    * Process Lead: `@github-userY`
    * Dev Crew: `@github-userZ`, ...
* **Summary of Work:** Created the repo. The Process Lead pushed the test
scaffold (Sync 1). The Dev Crew locally verified `pytest` and `flask run`
*before* pushing the `app.py` file. Completed a final round-trip sync
(Sync 2).
* **Evidence & Reflection:** In Part 5, the `Dev Crew` manually ran pytest
to get local evidence. This process relies on trust. What is the
professional risk of a workflow that relies on trusting every engineer to
remember to run tests, and how does this build the case for automation
(CI)?
```

After logging, commit and push this file. All other members must `git pull` to get the change.

Definition of Done (DoD) 🏁

In-Class DoD (Goal: 25 Minutes)

Your team's in-class work is "Done" when you can check the following:

- ☐ **Synchronization: All team members** have successfully completed **Part 7** and have the latest `CONTRIBUTIONS.md` file on their local machines.

Final DoD (Due: 11:59 PM Tonight)

The final Pull Request submission will be graded on the following "Done-Done" criteria:

- ☐ **Artifacts:** All required files (`app.py`, `tests/test_app.py`, `requirements.txt`, `.gitignore`) are present and correct.
- ☐ **Functionality:** The `Dev Crew` has confirmed that `pytest` passes and `flask run` works. (This is evidenced by the `Summary of Work` in the log).
- ☐ **Process:** `CONTRIBUTIONS.md` is updated with all roles and a thoughtful reflection.
- ☐ **Submission:** A Pull Request is open and correctly configured (see below).

Submission (Due: 11:59 PM Tonight)

This is a two-part submission.

1. **In-Class (Synchronization):** Your TA will verify your team's synchronization status at the end of the ICE.
2. **Homework (Pull Request): One team member** (e.g., the Process Lead) is responsible for opening the final Pull Request after class.
    - **Open Pull Request:** Open a new PR to merge your feature branch (`ice1-repo-setup`) into `main`.
    - **Title:** `ICE 1: MoJ Kickoff - Repo & Round-Trip Sync`
    - **Reviewer:** Assign your **Team TA** as a "Reviewer."
    - **Submit to Canvas:** Submit the URL of the Pull Request to the Canvas assignment.

TA Grading Rubric (10 Points)

The final PR (due 11:59 PM) will be graded using this rubric.

Criteria	Points	Description
Artifacts	4 pts	All required files (`app.py`, `tests/test_app.py`, `requirements.txt`, `.gitignore`) are present in the PR and are correct.
Process Log	3 pts	`CONTRIBUTIONS.md` is present, complete, and includes a thoughtful answer to the reflection question.
PR Hygiene	3 pts	The PR is titled correctly (`ICE 1: ...`) and has the **Team TA** assigned as a "Reviewer."
Total	**10 pts**	