Directions for CS494: Edge Computing Systems

Spring 2026

Standard Development and Submission

Below are requirements that apply to all assignments unless an assignment README states otherwise. If there is a conflict, the assignment README takes priority.

1. **Work in a separate branch:** Create a `development` branch and do your work there. You may create it using the command line, your IDE, or GitHub. All commits intended for submission must be on `development`.
2. **Regular commits:** Commit regularly and push at meaningful milestones to maintain a clear history and keep backups on GitHub.
3. **Submission:** Submit by opening a pull request from `development` into `main`. Do not merge the pull request unless explicitly instructed.
4. **Deadline:** Refer to the class schedule or assignment README. If not specified, the default due time is 11:59 PM on the due date.
5. **Late work:** See the syllabus for the late work policy.

Requirements for All Submissions

Certification and File Headers

All submissions must include a certification statement in the reflection or another location specified by the assignment README. In addition, include the following header comment in any new source file you create and in any starter file you substantially modify.

```
/*
  <Your Name>
  <Assignment>
  <Date>

  I certify that this is my work and, where appropriate, an extension of
  the starter code provided for the assignment.
*/
```

Build System Requirements

A *Makefile* may be used to build compiled components or to provide convenient run or validation targets for Python-based workflows, unless otherwise specified.

At a minimum, when a Makefile is required, provide these targets:

- `all`: builds or validates the full project
- `clean`: removes generated artifacts
- a program or task target (for example `a01` or `run`)

Running `make clean` should not error even if there is nothing to remove.

Python Environment and Dependency Requirements

For assignments that use Python, your repository must include a `requirements.txt` file or another dependency specification format stated in the assignment README. Your code must run in a clean virtual environment using only the listed dependencies.

- **Virtual environment:** Use a virtual environment such as `venv` to isolate dependencies from system Python.
- **Pinned dependencies:** List required third-party packages in `requirements.txt`. If versions affect correctness or grading, pin versions explicitly.
- **Reproducible setup:** A grader should be able to create a fresh environment, run `pip install -r requirements.txt`, and execute your program without additional manual steps.
- **Do not commit environments:** Do not commit `.venv` directories or installed packages. Use `.gitignore` as needed.

Reflection Requirements

Unless an assignment states otherwise, include a reflection paragraph in a text file named after the assignment (for example `a00.txt`). The file must be committed to the repository. The reflection should be clearly written, using proper grammar and spelling, and should provide meaningful insight into your experience with the assignment. Address what you learned, challenges you encountered, unexpected issues or discoveries, and what you would do differently in the future.

Documentation and Code Quality

1. **Consistency matters:** Adopt a published style guide and apply it consistently throughout the assignment. For example, the Google C++ Style Guide for C++ and PEP 8 for Python.
2. **Tooling requirements override preferences:** If an assignment or autograder requires specific filenames or extensions (for example .cc, .h, or .py), follow those requirements.
3. **Document your work:** Use clear function and file-level comments, and make intent obvious. Prioritize readability over cleverness.

Additional Resources

1. **Git Commands Help Sheet:** A quick reference to essential Git commands. It helps manage your code and changes efficiently.
2. **Learning Git and GitHub:** A collection of resources curated by GitHub to help you understand how to use Git and GitHub effectively for version control and collaboration.
3. **Makefile:** A very nice guide to the making of Makefiles.
4. **Python Virtual Environments:** A concise reference to creating and using virtual environments for reproducible Python workflows.