This folder contains a number of scripts which are used as part of the PyTorch build process. This directory also doubles as a Python module hierarchy (thus the `__init__.py`).

Overview

Modern infrastructure:

- autograd - Code generation for autograd. This includes definitions of all our derivatives.
- jit - Code generation for JIT
- shared - Generic infrastructure that scripts in tools may find useful.
    - module_loader.py - Makes it easier to import arbitrary Python files in a script, without having to add them to the PYTHONPATH first.

Build system pieces:

- setup_helpers - Helper code for searching for third-party dependencies on the user system.
- build_pytorch_libs.py - cross-platform script that builds all of the constituent libraries of PyTorch, but not the PyTorch Python extension itself.
- build_libtorch.py - Script for building libtorch, a standalone C++ library without Python support. This build script is tested in CI.
- fast_nvcc - Mostly-transparent wrapper over nvcc that parallelizes compilation when used to build CUDA files for multiple architectures at once.
    - fast_nvcc.py - Python script, entrypoint to the fast nvcc wrapper.

Developer tools which you might find useful:

- linter/clang_tidy - Script for running clang-tidy on lines of your script which you changed.
- extract_scripts.py - Extract scripts from `.github/workflows/*.yml` into a specified dir, on which linters such as linter/run_shellcheck.sh can be run. Assumes that every `run` script has `shell: bash` unless a different shell is explicitly listed on that specific step (so `defaults` doesn't currently work), but also has some rules for other situations such as actions/github-script. Exits with nonzero status if any of the extracted scripts contain GitHub Actions expressions: `${{<expression> }}`
- git_add_generated_dirs.sh and git_reset_generated_dirs.sh - Use this to force add generated files to your Git index, so that you can conveniently run diffs on them when working on code-generation. (See also generated_dirs.txt which specifies the list of directories with generated files.)
- linter/mypy_wrapper.py - Run `mypy` on a single file using the appropriate subset of our `mypy*.ini` configs.
- linter/run_shellcheck.sh - Find `*.sh` files (recursively) in the directories specified as arguments, and run ShellCheck on all of them.
- stats/test_history.py - Query S3 to display history of a single test across multiple jobs over time.
- linter/trailing_newlines.py - Take names of UTF-8 files from stdin, print names of nonempty files whose contents don't end in exactly one trailing newline, exit with status 1 if no output printed or 0 if some filenames were printed.
- linter/translate_annotations.py - Read Flake8 or clang-tidy warnings (according to a `--regex`) from a `--file`, convert to the JSON format accepted by pytorch/add-annotations-github-action, and translate line numbers from `HEAD` back in time to the given `--commit` by running `git diff-index --unified=0` appropriately.
- vscode_settings.py - Merge `.vscode/settings_recommended.json` into your workspace-local `.vscode/settings.json`, preferring the former in case of conflicts but otherwise preserving the latter as much as possible.

Important if you want to run on AMD GPU:

- amd_build - HIPify scripts, for transpiling CUDA into AMD HIP. Right now, PyTorch and Caffe2 share logic for how to do this transpilation, but have separate entry-points for transpiling either PyTorch or Caffe2 code.
    - build_amd.py - Top-level entry point for HIPifying our codebase.

Tools which are only situationally useful:

- docker - Dockerfile for running (but not developing) PyTorch, using the official conda binary distribution. Context: https://github.com/pytorch/pytorch/issues/1619
- download_mnist.py - Download the MNIST dataset; this is necessary if you want to run the C++ API tests.
- run-clang-tidy-in-ci.sh - Responsible for checking that C++ code is clang-tidy clean in CI on Travis