Tools

Documentation for various tooling in support of Deno development.

format.js

This script will format the code (currently using dprint, rustfmt). It is a prerequisite to run this before code check in.

To run formatting:

```
deno run --allow-read --allow-write --allow-run --unstable ./tools/format.js
```

lint.js

This script will lint the code base (currently using dlint, clippy). It is a prerequisite to run this before code check in.

To run linting:

```
deno run --allow-read --allow-write --allow-run --unstable ./tools/lint.js
```

Tip: You can also use cargo to run the current or pending build of the deno executable

```
cargo run -- run --allow-read --allow-write --allow-run --unstable ./tools/<script>
```

flamebench.js

`flamebench.js` facilitates profiling and generating flamegraphs from benchmarks.

General usage:

```
❯ ./tools/flamebench.js
flamebench <bench_name> [bench_filter]

Available benches:
op_baseline
ser
de
```

To profile the `op_baseline` bench, run `./tools/flamebench.js op_baseline`, this will run all 3 benches in `op_baseline.

Often when profiling/optimizing, you'll want to focus on a specific sub-bench, `flamebench` supports a bench/test filter arg like the regular cargo commands. So you can simply run `./tools/flamebench.js op_baseline bench_op_async` or `./tools/flamebench.js op_baseline bench_op_nop` to profile specific benches.

Tip: the `[bench_filter]` argument doesn't have to be an exact bench name, you can use a shorthand or a partial match to profile a group of benches, e.g: `./tools/flamebench.js de v8`

wgpu_sync.js

`wgpu_sync.js` streamlines updating `deno_webgpu` from gfx-rs/wgpu.

It essentially vendors the `deno_webgpu` tree with a few minor patches applied on top, somewhat similar to `git subtree`.

1. Update `COMMIT` or `V_WGPU` in `./tools/wgpu_sync.js`
2. Run `./tools/wgpu_sync.js`
3. Double check changes, possibly patch
4. Commit & send a PR with the updates