Layout files in git

This page answers: *which of an analog project's files can git compare and merge, which can't, and what does that change about committing, reviewing and resolving conflicts?*

Git in one page describes git working on text, line by line. An analog repository isn't all text: its main product, the GDS, is a binary file, and the files around it are a mix of sources you edit and outputs a tool writes. Which file is which tells you what goes into a commit, what a reviewer can read, and what to do when two people's changes collide. You'll resolve a conflict like that in Lab 04, and another in Lab 05.

Sources and outputs

File	Text or binary	Made by	Commit it?
`info.yaml`, `docs/info.md`, `src/project.v`	text	you	yes
`layout/init.tcl`, `layout/wires.tcl`	text	you	yes: they're the layout's source
`layout/tt_analog_1x2.def`	text	Tiny Tapeout	yes, with `UPSTREAM.md` saying where it's from
`layout/cells/*.mag`	text, in Magic's own format	Magic, from someone's drawing	yes: a hand-drawn cell has no other source
`gds/<top_module>.gds`	binary	`make` (Magic)	yes, because the `gds` workflow reads it from the repository
`lef/<top_module>.lef`	text	`make` (Magic)	yes, for the same reason
`layout/build/`, `tt_submission/`, Magic's `.ext` files, core dumps	mixed	`make`, `make precheck`	no: `.gitignore` keeps them out

The usual rule in software is to commit sources and leave out anything a build makes. The analog template breaks it on purpose: its `gds` workflow doesn't build anything, so the GDS and LEF have to be in the repository for it to check. The rule that's left is to commit them together with the change to the sources that made them, in the same commit, so that every commit's GDS is the layout of that commit's sources. Lab 05's history has a commit called "Keep build output out of git" that removed `gds/` from the repository, and the next one had to put it back, because the workflow reads it from there.

`.mag` files sit in between. Magic writes them, but for a cell drawn by hand in Magic's editor they are the only source there is, and they're text, so git can compare them. A reviewer still can't do much with a diff of one: `layout/cells/inv_bob.mag` is 116 lines of rectangles.

What a diff of a GDS shows

git treats a file as binary when it contains bytes that aren't text, and then shows only that it changed. This is from a rehearsal of Lab 04, a student claiming slot 4 of the sandbox:

```
gds/tt_um_ic102a_sandbox.gds | Bin 19716 -> 24770 bytes
info.yaml                    |   4 +-
layout/cells/inv_bob.mag     | 116 ++++++++++++++++++++++++++++++++++++++++++++++
```

```
layout/slots.tcl             |   1 +
lef/tt_um_ic102a_sandbox.lef |  57 ++++++++++++-----------
```

Five lines of that are readable, in `slots.tcl` and `info.yaml`. The GDS shows up only as a change in size. The LEF is text, but Magic generates it, and its 57 changed lines are Magic's bookkeeping. So a reviewer has to judge a layout change from other evidence:

- the source files that built it, which are text;
- the `gds_render` artifact, a picture of the layout;
- precheck's table and its `precheck_reports` artifact;
- the author's LVS result, which no workflow produces, so it has to be in the description.

Pull requests and review makes the same point for reviewers in general, and the sandbox's `REVIEWING.md` turns it into a checklist.

Two more things follow from the GDS being binary. First, every commit that changes it adds another whole version of it to the history, and every clone downloads all of them. Ours are about 20 KB, which is nothing, but a large layout is worth committing only when it has really changed. Git LFS, which keeps large files outside the history, isn't an option for a Tiny Tapeout project, because GitHub's documentation says a template "cannot include files stored using Git LFS", and a repository made from one starts without it. Second, Magic writes the time into every GDS it saves. Building the same sources twice gives two files that differ in 10 bytes, the dates in the file's headers, so `git status` shows your GDS as changed after any `make`, even when the layout didn't change. If you didn't mean to change the layout, `git checkout gds/ lef/` puts the committed files back.

Conflicts in binary files

When two branches change the same lines of a text file, a merge stops and shows you both versions between `<<<<<<<` and `>>>>>>>` markers. When they both change a binary file, there are no lines to show, and git says so:

```
warning: Cannot merge binary files: gds/tt_um_ic102a_sandbox.gds (HEAD vs. upstream/main)
Auto-merging gds/tt_um_ic102a_sandbox.gds
CONFLICT (content): Merge conflict in gds/tt_um_ic102a_sandbox.gds
```

git can offer you one side's file or the other's (`git checkout --ours` or `--theirs`), and both are wrong after a merge, because neither is the layout of the merged sources. In Lab 04′s rehearsal, one side's GDS had `bob`'s inverter and not `alice`'s, and the other had `alice`'s and not `bob`'s; either would have passed precheck, so nothing on GitHub would have caught the loss. The right answer is to merge the sources, which are text, and build the GDS again from the result. The LEF needs the same treatment even though it's text, because git merges it line by line into a file that describes no real layout: in the rehearsal it came out with three sets of conflict markers.

A revert is a merge too, run backwards: undoing a commit means applying its change in reverse to today's files. In Lab 05, reverting one commit stops on the GDS, which three later commits changed, while the one-line change to `wires.tcl` reverts cleanly. The fix is the same: run `make`, add the new GDS and LEF, and continue.

So, in a layout repository:

1. Resolve conflicts in the text sources by hand.
2. Don't resolve conflicts in `gds/` or `lef/`; rebuild them with `make`.
3. Check the result with `make lvs` and `make precheck` before you commit it, because a merge can produce a layout that neither side tested.

Check yourself

1. You change one coordinate in `wires.tcl`, run `make`, and commit only `wires.tcl`. What does the next `gds` run check?

   Answer: A1, in Answers at the end.

2. After a merge, `make` fails and `info.yaml` won't parse. What's the likely cause?

   Answer: A2, in Answers at the end.

Answers

A1. Check yourself

The GDS from the previous commit, because that's the one in the repository. The run tests a layout that doesn't match the sources in the same commit. Commit `gds/` and `lef/` with the change.

A2. Check yourself

Conflict markers left in `info.yaml`. The course's `make` says so when it can't read `top_module`, and lists the marker lines.