Lab 05: Find the commit that broke it

You'll take over the history of a small Tiny Tapeout project, an LED dimmer that three teammates wrote over five weeks. Its test passed at the commit they tagged `v1.0`, and it fails now, 27 commits later. You'll write a script that tells `git bisect run` correctly whether each commit is good, bad or can't be tested, use it to find the commit that broke the test, revert that commit on a branch, and get the revert through every workflow in a pull request on a repository of your own.

Package: `labs/lab-05-find-the-commit-that-broke-it/`

Before you start

- Read History as evidence.
- Finish Lab 03. This lab uses the test job's tools you installed in Lab 01, Part 1, your access token from Lab 01, Part 2, and the way you turned on Pages in Lab 01, Part 5.
- Every command runs in a terminal on the workbench desktop, where `git bisect run` can call the same `make test` as Lab 03. In each new terminal, run these first; the steps use `$L` for this lab's package and `$TT` for the test job's tools:

```
. /foss/designs/common/.designinit
L=/foss/designs/modules/ic102-d/labs/lab-05-find-the-commit-that-broke-it
TT=/foss/designs/modules/ic102-d/labs/common/tt-tools
```

The teammates are made up, and so is their project, which the course builds from Tiny Tapeout's template with a script. Every commit is an ordinary one, though: features, tests, documentation, two half-finished edits, a new file left out of the test's source list, and one tidy-up that went wrong.

This lab takes about an hour, including the 5 to 10 minutes each push takes on GitHub's runners.

Part 1: Clone the history

What does the test say now, and what did it say at `v1.0`?

1. Clone the history. The course ships it as a git bundle, a single file that holds every commit and tag, and `git clone` reads it like a repository:

   ```
   mkdir -p /foss/designs/projects && cd /foss/designs/projects
   git clone $L/data/history.bundle ttsky-ic102-d-bisect
   cd ttsky-ic102-d-bisect
   git config user.name "Your Name"
   git config user.email "ID+USERNAME@users.noreply.github.com"
   git config credential.helper 'cache --timeout=28800'
   ```

   Use the same no-reply address as in Lab 01. You should see `Cloning into 'ttsky-ic102-d-bisect'...`. The folder is named after the repository you'll push it to in Part 4.

2. Look at the history:

   ```
   git log --oneline | wc -l
   git log --oneline v1.0..main | wc -l
   git log --oneline -3
   ```

   You should see `36`, then `27`, then:

```
00c3d87 test: name the LED's pin in failure messages
35f4b2e info.yaml: a longer description
14d59b6 test: run fade mode for 100 steps
```

The hashes in your clone are the same as the ones on this page, because the course built each commit with a fixed author, date and message.

3. Run the test on `main`, as the test job would:

```
make -C "$TT" test PROJECT="$PWD"
```

You should see three of seven tests fail:

```
AssertionError: with duty 0 the LED (uo_out[7]) was on for 8 of 2048 cycles, expected 0
AssertionError: with duty 64 the LED (uo_out[7]) was on for 260 of 1024 cycles, expected 256
AssertionError: with duty 255 the LED (uo_out[7]) was on for 1024 of 1024 cycles, expected
1020
```

and `** TESTS=7 PASS=4 FAIL=3 SKIP=0` in the summary table, followed by `make: *** [Makefile:42: test] Error 2`.

4. Before you run it, write down whether the test passes at `v1.0`. Then check:

```
git switch --detach v1.0
make -C "$TT" test PROJECT="$PWD"
git switch main
```

You should see `** TESTS=3 PASS=3 FAIL=0 SKIP=0`. The test had only three cases then, all of them about the PWM, and all three pass.

5. Somewhere in the 27 commits between `v1.0` and `main`, the PWM broke. How many tests will `git bisect` need to find the commit?

Answer: A1, in Answers at the end.

You should see: three PWM failures on `main`, and a pass at `v1.0`.

Part 2: The bisect script

What should the script say about a commit whose design doesn't compile?

1. Copy the starting script to the top of your clone, and read it:

```
cp $L/bisect-template.sh bisect.sh
```

It runs the test job's steps with `run-test.sh`, the script behind Lab 03's `make test`, and then always exits 0. The comment at the top lists the three answers `git bisect run` understands: 0 for good, 1 to 127 except 125 for bad, and 125 for "can't test this one". `bisect.sh` isn't tracked by git, so checking out old commits leaves it alone.

2. Why can't the script simply exit with `run-test.sh`'s status? Try it on `main`, where a test fails, and on `cbba6a3`, a commit from the middle of the history:

```
for c in main cbba6a3; do
  git checkout -q $c; rm -f test/results.xml
  "$TT/run-test.sh" . > /dev/null 2>&1
  echo "$c: exit $?; $(ls test/results.xml 2>&1)"
done
git checkout -q main
```

You should see:

```
main: exit 2; test/results.xml
cbba6a3: exit 2; ls: cannot access 'test/results.xml': No such file or directory
```

Both exit with 2. With cocotb 2.x, `make` runs cocotb's own results check at the end of the simulation, so it stops with `Error 2` when a test fails, and it stops with the same status when the design doesn't compile. The difference shows in `test/results.xml`: cocotb writes it whenever the simulation runs, and never when the compile fails. (This is also why the script deletes `results.xml` before each run. An old one, left from the previous commit, would make a commit that never ran look as if it had.)

3. Replace the last line of `bisect.sh` (`exit 0`, under the TODO) with the rule the History as evidence script uses for `protocol-emulator`, where a missing `results.xml` counts as bad:

```
if [ ! -f test/results.xml ]; then
    exit 1
fi
if grep -q "<failure" test/results.xml; then
    exit 1
fi
exit 0
```

4. Run the bisect between the last known good commit and `main`:

```
git bisect start main v1.0
git bisect run ./bisect.sh
```

It takes about ten seconds. You should see it test five commits and end with:

```
85133d69064177106e14d05cf0e7b97a7add19f5 is the first bad commit
commit 85133d69064177106e14d05cf0e7b97a7add19f5
Author: Sam Rivera <sam.rivera@example.com>
Date:   Thu Oct 15 20:25:00 2026 -0400

    Move the seven-segment table into seg7_table.v

 info.yaml        |  1 +
 src/seg7.v       | 24 +++--------------------
 src/seg7_table.v | 35 ++++++++++++++++++++++++++++++++++++
 3 files changed, 39 insertions(+), 21 deletions(-)
 create mode 100644 src/seg7_table.v
bisect found first bad commit
```

5. Is `85133d6` the commit that broke the PWM? The script kept a log of each commit it tested; read this one, and the next commit's message:

```
grep error /tmp/bisect-85133d6.log
git log --oneline -1 ae2c625
```

   Answer: A2, in Answers at the end.

6. End this bisect, change the first `exit 1` in `bisect.sh` (the one under `if [ ! -f test/results.xml ]`) to `exit 125`, and run it again:

```
git bisect reset
git bisect start main v1.0
git bisect run ./bisect.sh
git bisect log
```

   You should see the first commit it checks out, `cbba6a3 WIP: preset brightness levels on uio_in[2:1]`, reported as skipped in the log, five commits tested in all, and a different first bad commit. Write down its hash and message before you open the answer.

   Answer: the bisect, from git bisect log: A3, in Answers at the end.

7. Read the commit, then end the bisect:

```
git show <the first bad commit>
git bisect reset
```

   Its message says it tidied up the comparator and its comments. Which line changed what the circuit does, and why does that turn the LED on for exactly 8 of 2048 cycles at duty 0?

   Answer: A4, in Answers at the end.

8. Check your script against three commits whose answers are known:

```
python3 $L/check.py --offline
```

   You should see, after a few seconds (5 s in our run):

```
Your bisect.sh, run at three commits
  4e04d77  v1.0, test passes           exit 0    ok
  cbba6a3  preset WIP, doesn't compile  exit 125  ok
  00c3d87  last commit, test fails      exit 1    ok
```

   followed by `no commit made by git revert on this branch yet`. You'll make that commit in Part 3.

You should see: `git bisect` name a commit whose diff changes the PWM, and three `ok` lines from the check.

Part 3: The revert

Can you undo that one commit without undoing the seven commits that came after it?

1. Before you run anything, look at what the seven later commits touched (`git log --stat --oneline <the first bad commit>..main`). Write down whether you expect `git revert` to apply cleanly.

   Answer: A5, in Answers at the end.

2. Make a branch for the fix and revert the commit on it:

```
git switch -c revert-pwm-compare main
git revert --no-edit <the first bad commit>
git log -1 --format=%B
```

   You should see `1 file changed, 4 insertions(+), 4 deletions(-)` and a message that ends with `This reverts commit <its full hash>.` The new commit's own hash will differ from anyone else's, because it carries your name and the time.

3. Run the test again:

```
make -C "$TT" test PROJECT="$PWD"
```

   You should see all seven pass, among them `test.test_duty_0_is_off PASS 205250.00` and `** TESTS=7 PASS=7 FAIL=0 SKIP=0`. The fade, preset and display tests added after the bad commit still pass, because the revert undid only the comparator.

4. Commit your script on the branch, so that the pull request carries the evidence:

```
git add bisect.sh
git commit -m "Add the bisect script that found the PWM regression"
```

You should see: seven passing tests on your branch, with the later features still in place.

Part 4: The pull request

Does GitHub's runner agree with your workbench?

1. On GitHub, create a new public repository called `ttsky-ic102-d-bisect` in your own account. Leave it empty, as in Lab 04, Part 1: no README, no `.gitignore` and no licence, because the history you push already has its own first commit.

2. Point your clone at it and push `main` and the tag. The push sends files under `.github/workflows/`, which GitHub accepts only from a token with the Workflows permission you gave yours in Lab 01, Part 2:

```
git remote set-url origin https://github.com/YOUR-USERNAME/ttsky-ic102-d-bisect.git
git push -u origin main --tags
```

3. Before you look at the Actions tab, predict the result of each job on `main`: `docs`, `test`, `gds`, `precheck`, `gl_test` and `viewer`.

   Answer: A6, in Answers at the end.

   You'll also see the three workflows run a second time, on `v1.0`. Tiny Tapeout's workflows run on every push, and pushing a tag counts as one. On `v1.0` only `viewer` fails, with the same

   404.

4. Turn Pages on for `ttsky-ic102-d-bisect` the way you did in Lab 01, Part 5. You don't need **Run workflow** this time, because the push in the next step starts a new run.

5. Push your branch and open a pull request from `revert-pwm-compare` into `main`:

```
git push -u origin revert-pwm-compare
```

GitHub prints a link for creating the pull request. In the description, name the commit you reverted and why it was wrong, say how you found it, and paste the output of `git bisect log` from Part 2. If an AI tool helped you write the script or the description, say so and say what you checked yourself, as Pull requests and review asks.

6. Wait for the runs on your branch to finish (about 5 to 10 minutes), then run the full check:

```
python3 $L/check.py
```

It repeats Part 2's three runs and then reads your branch's latest commit from GitHub. In the course's practice repository it printed:

```
Checking /foss/designs/projects/ttsky-ic102-d-bisect

Your bisect.sh, run at three commits
  4e04d77  v1.0, test passes           exit 0    ok
  cbba6a3  preset WIP, doesn't compile  exit 125  ok
  00c3d87  last commit, test fails      exit 1    ok

Your revert
  reverts d18e316, the commit that broke the test

GitHub: vproHacks/ic102-test-d at 6b2ae2a
  docs      success
  test      success
  gds       success
  precheck  success
  gl_test   success
  viewer    success

VERDICT: PASS
```

`viewer` passes on a branch too: it publishes the branch's layout to your project's page.

7. Merge the pull request. The repository is yours and has no rules, so you can merge it yourself. The merge is a push to `main`, so `main`'s workflows run again, now with the revert in them and Pages on. In the course's practice repository all six were green 5 min 45 s after the merge, `viewer` included.

You should see: all six jobs green on the pull request, `VERDICT: PASS`, and all six green on `main` after the merge.

Finishing

You're done when `check.py`, run on your `revert-pwm-compare` branch, prints `VERDICT: PASS` (your script gives the right answer at all three commits, your branch reverts the commit that broke the test, and all six jobs passed on the branch's latest commit), and every job is green on `main` after you've merged the pull request.

Questions:

1. Your script and the `protocol-emulator` script in History as evidence give opposite answers for a commit that doesn't compile. Can both be right?

   Answer: A7, in Answers at the end.

2. The `v1.0` tag saved you a step. What would you have had to do without it?

   Answer: A8, in Answers at the end.

3. The bad commit was made on Oct 25, and seven more commits followed before anyone noticed. In the team's real repositories the test workflow runs on every push. What does that suggest the made-up team was doing wrong?

   Answer: A9, in Answers at the end.

Submit: the link to your pull request.

Going further (optional)

When you already know which line is wrong, git can find the commit that touched it without running anything. `git log -G` lists the commits whose diffs add or remove a line matching a pattern:

```
git log --oneline -G "phase <" -- src/pwm.v
```

On the `revert-pwm-compare` branch of the course's practice repository it printed:

```
840c251 Revert "pwm: tidy up the comparator and its comments"
d18e316 pwm: tidy up the comparator and its comments
c6218cc Add an 8-bit PWM
```

From the bottom up, those are the commit that wrote the comparison, the commit that changed it, and the revert that changed it back, which has a hash of its own in your clone. `git bisect` is for the more common case, where you know only that the test fails.

This is the last row of the Course path.

Answers

A1. Part 1: Clone the history

About five. Each test halves the number of suspects, and $2^5 = 32 \geq 27$. One commit that can't be tested may add a test or two, as you'll see in Part 2.

A2. Part 2: The bisect script

No. The log has a line ending in `src/seg7.v:15: error: Unknown module type: seg7_table`: the commit added a new file to `info.yaml` but not to `PROJECT_SOURCES` in `test/Makefile`, so the design didn't compile and the test never ran. The very next commit, `ae2c625 test/Makefile: add seg7_table.v to PROJECT_SOURCES`, fixed that, and nothing about the PWM changed. The script reported a commit that couldn't be tested at all as bad, and `git bisect` goes by whatever the script reports.

The same rule was right for `protocol-emulator`, because there the failure being hunted *was* a compile error (a module the testbench could no longer find). Here the failure being hunted is a wrong value from the PWM, and a commit that doesn't compile tells you nothing about it.

A3. Part 2: The bisect script (Answer: the bisect, from git bisect log)

```
# bad: [00c3d87894d0c8bce7a127e029cd9f6cbbb36da2] test: name the LED's pin in failure messages
# good: [4e04d775e153a5c2526f9687f9ef83a185f62b7d] docs: say how the dimmer works
git bisect start 'main' 'v1.0'
#  skip:  [cbba6a3831a677c6552cf449fd3ae56728bf2565]  WIP:  preset  brightness  levels  on
```

```
uio_in[2:1]
git bisect skip cbba6a3831a677c6552cf449fd3ae56728bf2565
# good: [921b40562b8fbf021b317bc475c61ee5f525c1b7] presets: finish the case statement
git bisect good 921b40562b8fbf021b317bc475c61ee5f525c1b7
# bad: [d18e3169a759beec08f58e0f6327310941a607de] pwm: tidy up the comparator and its comments
git bisect bad d18e3169a759beec08f58e0f6327310941a607de
# good: [3ad66712b887c83d196dd5e4038357ec66c468b8] test: the three presets
git bisect good 3ad66712b887c83d196dd5e4038357ec66c468b8
# good: [60651505fa672660194fbb3cb412ff63d39e56f9] test: run the testbench clock at 10 MHz,
to match clock_hz
git bisect good 60651505fa672660194fbb3cb412ff63d39e56f9
# first bad commit: [d18e3169a759beec08f58e0f6327310941a607de] pwm: tidy up the comparator
and its comments
```

A4. Part 2: The bisect script

`assign out = (phase < duty);` became `assign out = (phase <= duty);`. The phase counts 0 to 255, once per PWM period, so with duty 0 the new comparison is true for one step in every period, when the phase is 0. A step lasts four clock cycles (the prescaler divides by 4), and the test watches two periods: $2 \times 4 = 8$ cycles. The same extra step gives 260 instead of 256 cycles at duty 64, and turns duty 255 into always on. The four changed comment lines around it make the diff look like a comment edit, which is how it got past its author.

A5. Part 3: The revert

It should. None of the later commits touch `src/pwm.v`, so the lines the revert puts back haven't moved, and git can apply the reverse of the old diff directly.

A6. Part 4: The pull request

`docs`, `gds` and `precheck` should pass, because the files they check are fine. `test` fails on the three PWM cases, and so does `gl_test`, which runs the same test on the gate-level netlist. `viewer` fails with the 404 from Lab 01, because a new repository has Pages off.

That's what the course's practice repository got. `test` and `gl_test` both stopped at `AssertionError: with duty 0 the LED (uo_out[7]) was on for 8 of 2048 cycles, expected 0`, the first of the three failures you saw in Part 1, and `viewer` at `Error: Failed to create deployment (status: 404)`.

A7. Finishing

Yes, because they hunt different failures. In `protocol-emulator` the test broke *because* the testbench stopped compiling, so a commit that doesn't compile shows the bug. In the dimmer the bug is a wrong value from a design that compiles, so a commit that doesn't compile can't show it either way, and 125 tells git to look at a neighbour instead. So decide what you're hunting before you decide what the script says about a commit that doesn't compile.

A8. Finishing

Find a good commit yourself before `git bisect start` could begin, by checking out older commits and running the test until one passed. A tag on a commit that everyone saw green is a known-good starting point for anyone who comes later, which is why History as evidence suggests marking one.

A9. Finishing

They weren't looking at their runs. On GitHub, the `test` workflow would have gone red on the push that included `d18e316`, and on every push after it. A red cross only stays red for a week when nobody is reading it, and that's how `hart-1`'s `docs` and `gds` stayed red for four weeks.