The publish loop

The explicit local-to-public boundary, checks, and why a merge is not a deploy.

JWOGRADY REV 1.1

CONTENTS

1. Where things publish
2. Local and public
3. The loop
4. By hand, without the script
5. Check before pushing, and after
6. Capture without publishing
7. Review, then explicitly make a note public

Three commands and one rule. The rule is that a clean build is not a deploy.

1. Where things publish

`master`. It is GitHub's default branch **and** the branch Netlify builds, and those two being the same is deliberate.

They disagreed once, on 2026-09-22, and it cost a day: four entries were merged onto `release/pub-v0.1.0`, every build was clean, the publish script reported success four times, and nothing reached the site. Netlify's production branch lives in its console and no key in `netlify.toml` sets it, so nothing in the repo could have told you. There is a comment at the top of `netlify.toml` now that says so. If the console setting ever changes, change that comment in the same sitting.

2. Local and public

Every page is in one of two states, and **local is the default**.

STATE	FRONT MATTER	BUILT FOR
public	`draft = false`	www.jwogrady.com ‹https://www.jwogrady.com›, previews, and local
local	anything else	`http://localhost:8080/` only

A cascade in `hugo.toml` gives every page `draft = true` unless it says otherwise. A page that leav the key out is local, not public, so nothing reaches the site by omission. Going public is always an explicit `draft = false`, and `story.sh publish` is what writes it.

Netlify never passes `-D`, so it builds public pages only. Locally, one command builds everything an serves it from the `jwogrady` wslc container:

```
bin/local-preview               # build the working tree, local + public, and serve it
bin/local-preview --build-only  # same build, into ./public-local
```

It builds the working tree as it stands, uncommitted edits included, with drafts and future-dated pages. It also folds in any entry still sitting on an unmerged `story/*` or `wip/*` branch. Its `robots.txt` says `Disallow: /`.

3. The loop

The lifecycle is a script, so the git mechanics are not re-improvised per entry:

```
.claude/skills/disbatch/scripts/story.sh plan <slug> [title...]
.claude/skills/disbatch/scripts/story.sh check [slug]
.claude/skills/disbatch/scripts/story.sh publish <slug>
.claude/skills/disbatch/scripts/story.sh list
```

`plan` branches `story/<slug>`, assigns the next citation handle by scanning the existing entries, dates the entry a minute in the past, and scaffolds the six beats as comments.

The date matters: Hugo silently drops future-dated content. A stamp even a minute ahead produces 404 with no error and no warning.

`check` builds with and without drafts, then reports TODO markers, word count, figures, tags, and whether the page actually rendered. `renders NO` with a plausible-looking entry almost always means the date is in the future.

`publish` refuses an entry that still has TODO markers or no tags — the second would render and appear in no index, which has happened. Then it undrafts, verifies the page survives a production build, commits, merges `--no-ff` onto `master` and deletes the branch.

`list` is every entry with its handle, state and length, plus the tag counts and the number of distinc
counts — that number *is* how many hue bands the index can draw.

4. By hand, without the script

```
hugo new posts/what-i-learned-today.md      # or a bundle, below
hugo server -D                               # preview, drafts included
hugo --gc --panicOnWarning                   # what the host runs
```

For an entry with its own image, make it a **page bundle**, because that is what lets the theme resize and convert:

```
mkdir content/posts/my-entry
hugo new posts/my-entry/index.md
# images go in content/posts/my-entry/, beside index.md
```

The difference is shipping a 1.7 MB PNG or a 37 KB WebP.

5. Check before pushing, and after

```
hugo --buildDrafts        # does it build at all
hugo                      # does it build without drafts
```

The second one matters: an entry that renders under `-D` and vanishes in production is the most common way to publish nothing.

And then the part that is easy to skip, because the script has already said `published`:

```
curl -s -o /dev/null -w '%{http_code}\n' https://www.jwogrady.com/posts/<slug>/
```

A merge is not a deploy and a deploy is not a page. The only evidence that something published is th
live URL answering `200`.

6. Capture without publishing

Lab Notes do not need the six-beat article scaffold. Use the site-owned `bin/lab` tool to start with or
observation, a selected excerpt or an experiment. It always writes `draft = true`, never commits,
and never pushes. Capture does not imply approval.

```
bin/lab new first-observation --title 'A useful observation' --tag workflow
bin/lab from-markdown selected-conversation /path/to/selected-excerpt.md --tag workflow
bin/lab from-logseq selected-finding --block BLOCK-UUID --tag workflow
bin/lab from-notebook experiment /path/to/experiment.ipynb --tag experiments
```

Select a Logseq block explicitly; the tool does not read the journal, scan the graph, follow block references or export children. Add `--block` again for another block. It uses the existing local `logseq-api` helper, whose token is read from 1Password at use time. A block marked `private:: true` is refused. References to unselected blocks must be resolved or removed.

Assets are opt-in too: `--asset /path/to/screenshot.png` copies that file, not the surrounding folder. Add context and alt text before publication. Review excerpts for other people's private detail tokens and client data. The tool cannot make that judgment for you.

Notebook capture reads a saved `.ipynb` file; it never runs a kernel. Outputs are omitted by default. `--include-outputs` opts into stored plain text and PNG/JPEG output, not HTML, JavaScript or widgets. Review generated images, replace their placeholder alt text, and do not treat old outputs as fresh execution. Select the original notebook with `--asset` only if it is safe to offer as a download.

7. Review, then explicitly make a note public

```
bin/lab check first-observation
# Review the exact text AND assets corresponding to the printed hash.
bin/lab publish first-observation --approve-sha256 REVIEWED-HASH
```

`check` performs mechanical checks and prints a hash of the bundle. That is not human approval. `publish` requires that exact hash and changes only the selected note's draft flag. Any edit or asset change invalidates an older hash. This is a local operation, not a commit, push, merge or deployment. Use the normal reviewed Git release process afterward, and check the live URL. Do not run the older Journal `story.sh publish` over a dirty working tree: it stages all changes, not just the story.

Preview drafts on loopback only (`hugo server -D --bind 127.0.0.1 --port 1317`). The production build must run without `-D`. `/index.xml` is Journal-only; `/lab-notes/index.xml` is Lab Notes-only. Tags, projects and archives are shared indexes with explicit writing-type labels. There is no site search in the current theme; these indexes are the discovery paths.

Canonical: https://www.jwogrady.com/docs/publishing/ ‹https://www.jwogrady.com/docs/publishing/›