GOOD COMMITIZENSHIP

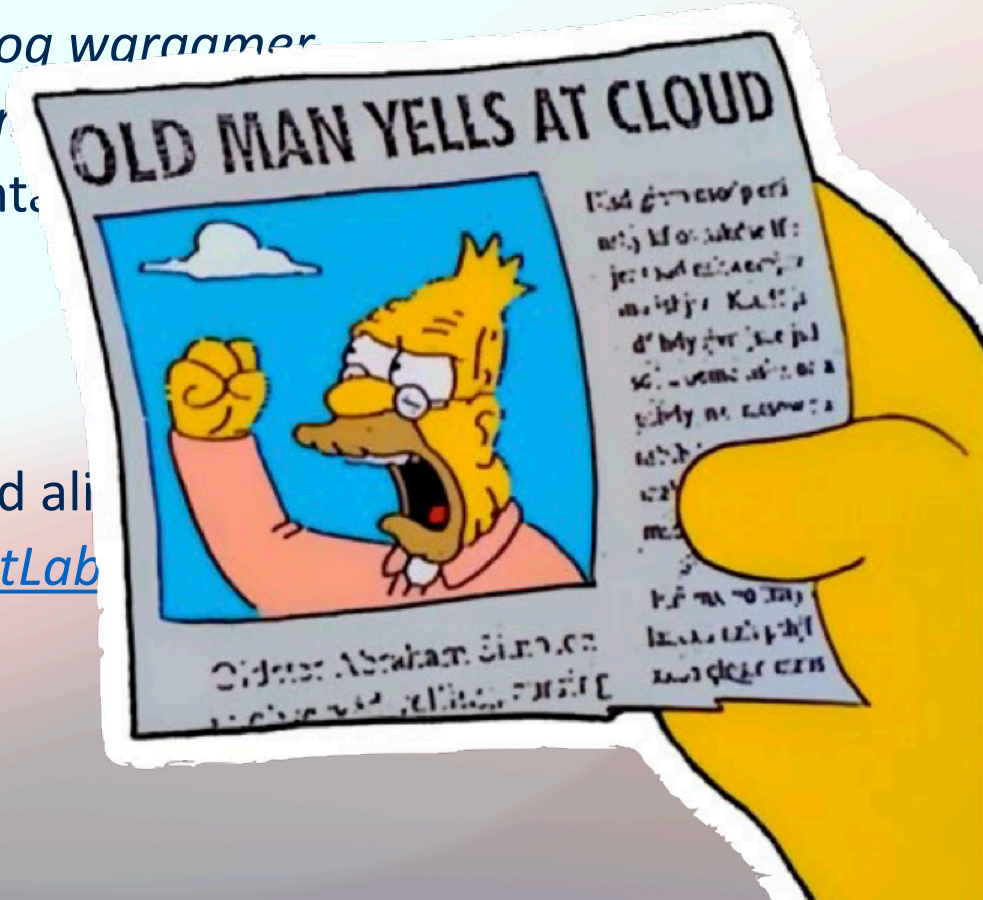
Agenda

- Introductions
- Funny Because It’s True
- How Do We Fix This?
  - Git Hooks
  - Some Obvious, but Important Notes
- Tooling
  - Husky
  - Conventional Commit Specification
    - The Standard and its variants
  - Commitlint (Commitizen, ...)
- Release and Version Automation
- Demo

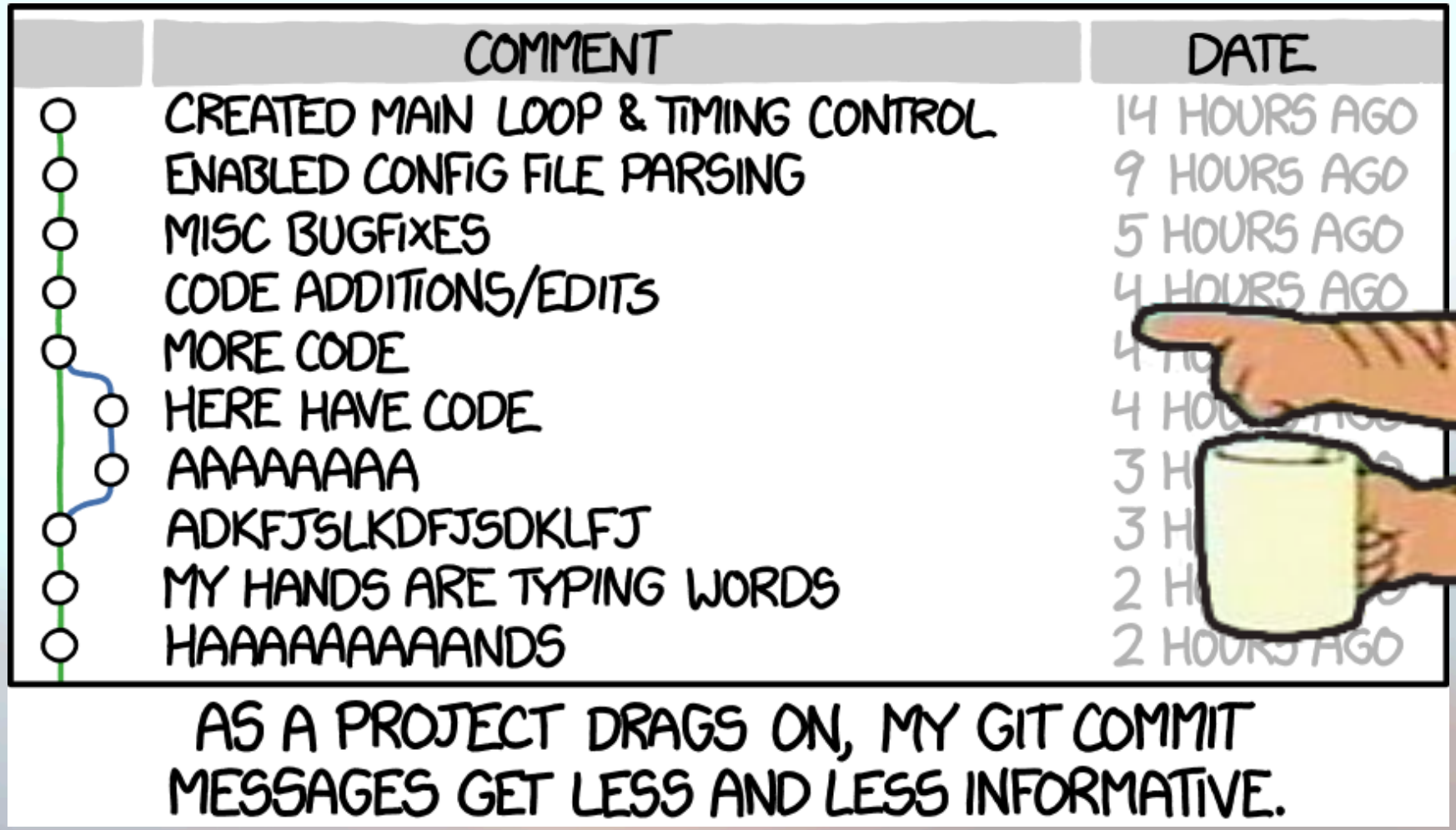
INTRODUCTIONS

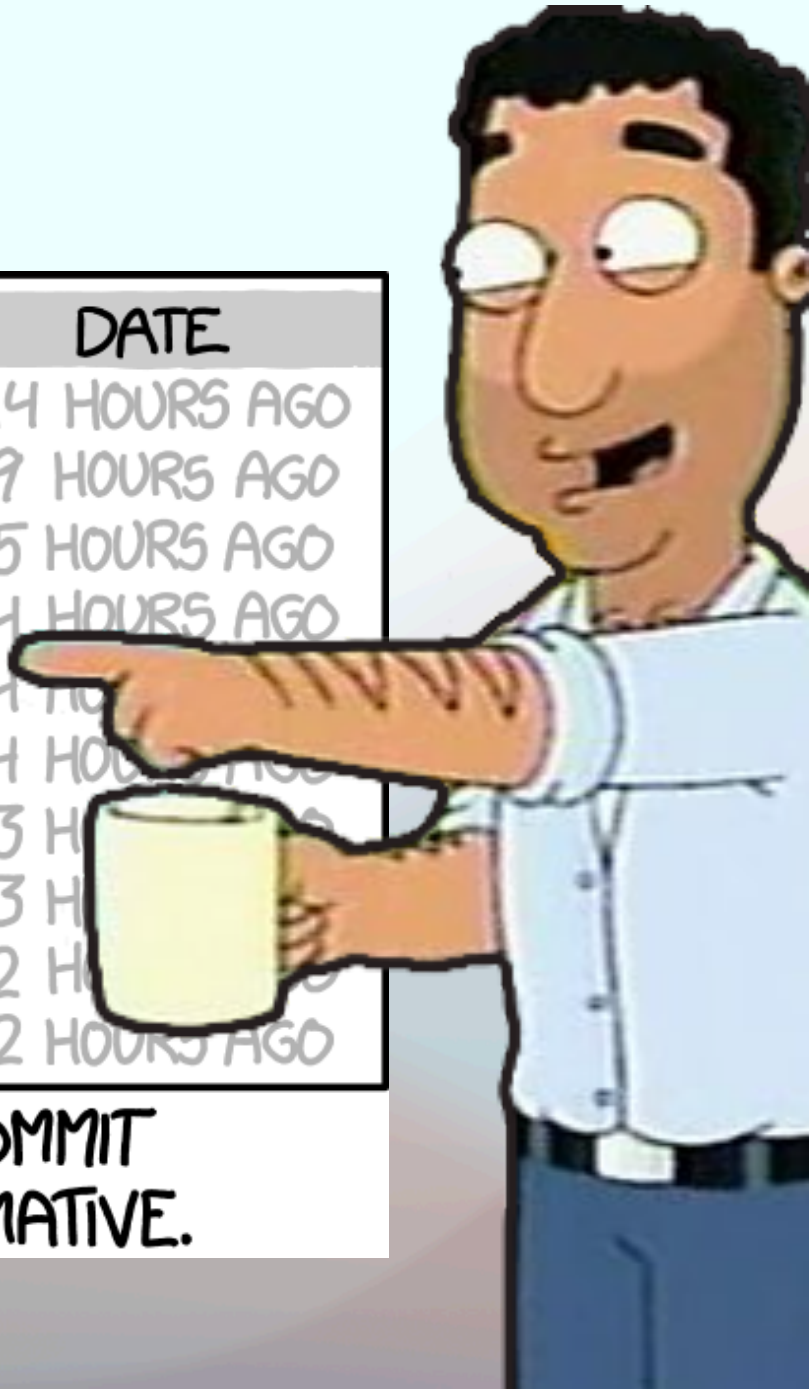
Introductions

- Timothy Stone
  - Some call me... *Tim*
  - Solution Architect, Capital One SBB Tech
  - *...a father, blogger, OSS committer, source control nerd, Java certified web developer, analog wargamer, home brewer, and lifelong TTRPG gamer*
  - Find me on LinkedIn, GitLab, GitHub, Unt
  - I'm opinionated
  - I will be using the command line, a lot.
    - Z-Shell + Oh My ZSH + git plugin
    - I'll try to vocalize actions and expand ali
  - All code available will be available on *GitLab*



It’s Funny Because It’s True





https://xkcd.com/1296

How Do We Fix This?

- Git Hooks

  “Like many other Version Control Systems, Git has a way to fire off custom scripts when certain important actions occur...

- Possibly obvious, but an important note
  - Client-side hooks are not shared
  - `.git/hooks` exists only on the developer’s clone...
  - `.git` *ignored in the binary and* cannot be committed
    - `git check-ignore -v .git`

TOOLING

Introducing Husky

- Modern, native git hooks made easy
  - “Native git hooks?”
  - Sets `core.hookspath` to `.husky`

```
npm install --save-dev husky
npx husky init
```

The `init` command simplifies setting up husky in a project. It creates a `pre-commit` script in `.husky/` and updates the `prepare` script in `package.json`.

```
git config --get core.hookspath
git add --all
git commit --verbose --all
```

- BAM! Project portable commit controls

Commit Standards

- Having project hooks is nice
- Commit standards are needed
  - What is the Standard?
    - Do you already have one?
    - Do you have time to write one?
    - Enter *Conventional Commits Specification 1.0.0*
  - Do you Force or Nudge Team Members?
    - Add controls and a dash of mentorship?
  - Do you Educate?
    - Slow and inconsistent

Commit Controls

- You've adopted standards...
- Enter Commitlint[1]
  - "Helps your team adhere to a commit convention"

```
% npm install --save-dev @commitlint/cli
% npm install --save-dev @commitlint/config-conventional
% echo "npx --no -- commitlint --edit \$1" > .husky/commit-msg
% chmod +x .husky/commit-msg[2]
% vi .commitlintrc.js
... Supports YAML, Typescript (.ts, .cts), JSON, CommonJS (.cjs),
and ECMA Modules (.mjs) too!
% npx commitlint --print-config
% git add --all
% git commit --verbose --all
```

1 There are others, e.g., commitizen. YMMV

2 In Husky v9, the executable bit is optional. The *nix executable bit is a committable change; ignored by native Windows; on Windows, but pipeline is *nix? `git --update-index --chmod=+x .husky/commit-msg`

Commit Communist!

- Push back happens.

```
git add --all
git commit --verbose --all --no-verify
```

  - `--no-verify` bypasses hooks by ignoring non-zero exit codes
- Do all my contributors need to use the Conventional Commits specification?
  - No
- Nudge and mentor, then... Squash
  - Merge leads can fixup commits
  - This becomes especially necessary when coupling commit messages to...
    - Pipelines and automated Semantic Versioning

FAQs

My Project Isn’t JavaScript...

- Git Hooks are the Honeybadger
  - They “Don’t Care”
- `node` and `npm` (`yarn`, yada yada) should be available
  - For developers
  - In the pipeline
- Java, Python, Rust, Go...
  - Add
    - `.commitlintrc.js`
    - `package.json`[1]
    - `package-lock.json`
  - Husky and Git Hooks take care of the rest

```
1 npm install # runs prepare script, sets core.hookspath
```

My CICD Build Targets...

- YMMV
- GitLab uses Job runners with artifacts and dotenv support
- Pseudo-pipeline descriptor…

```
semantic-release-info:      # job
   stage: .pre
   image: node:latest       # job runner
   script:
     - ...
     - semantic-release --dry-run  # writes dotenv to pass to release job
mvn-build:
  image: maven:3.8-eclipse-temurin-17
  script:
    ...
mvn-release:
  image: maven:3.8-eclipse-temurin-17
  script:
    - mvn release:prepare -DreleaseVersion=${SEMREL_INFO_NEXT_VERSION}"...
    - ...
```

My CICD Pipeline Makes Commits

- Common concern
- Pipelines may fail
- Options:
  - Configure CICD tooling, e.g.,

    `-DscmDevelopmentCommitComment="ci(foo): prepare..."`
    - Can be tedious and involve a lot of testing
  - Test executing context
    - `is-ci` to the rescue!
      - Supports 40+ CI platforms
    - `HUSKY=0`

```
npm install --save-dev is-ci
npm pkg set scripts.prepare="is-ci || husky"
```

Next Steps

- Integrate with Semantic Release
- Commit types inform Semantic Versioning
  - `fix` and `feat` — API relevant
    - patch and minor, respectively
    - `feat!` — major, reserved for breaking changes
      - Commit footers fully supported
      - `BREAKING CHANGE:`
        `<blank line>`
        `No more support for a feature we know you love.`
  - All other types noop in Semantic Versioning
    - Merge to trunk! All. Sprint. Long.
- Borrow...
  - To Be Continuous Templates and GitHub Actions
    - Free insight for your CICD tooling
    - Tread carefully though
- Git Template
  - `git config --add --global commit.template template`
  - Mine

Demo

- A walk through of adding commitlint to a project.

QUESTIONS?