

# git hooks

Task automation for your repos

*Jordan Duabe* |  @jordan@jduabe.dev  @j4ckofalltrades

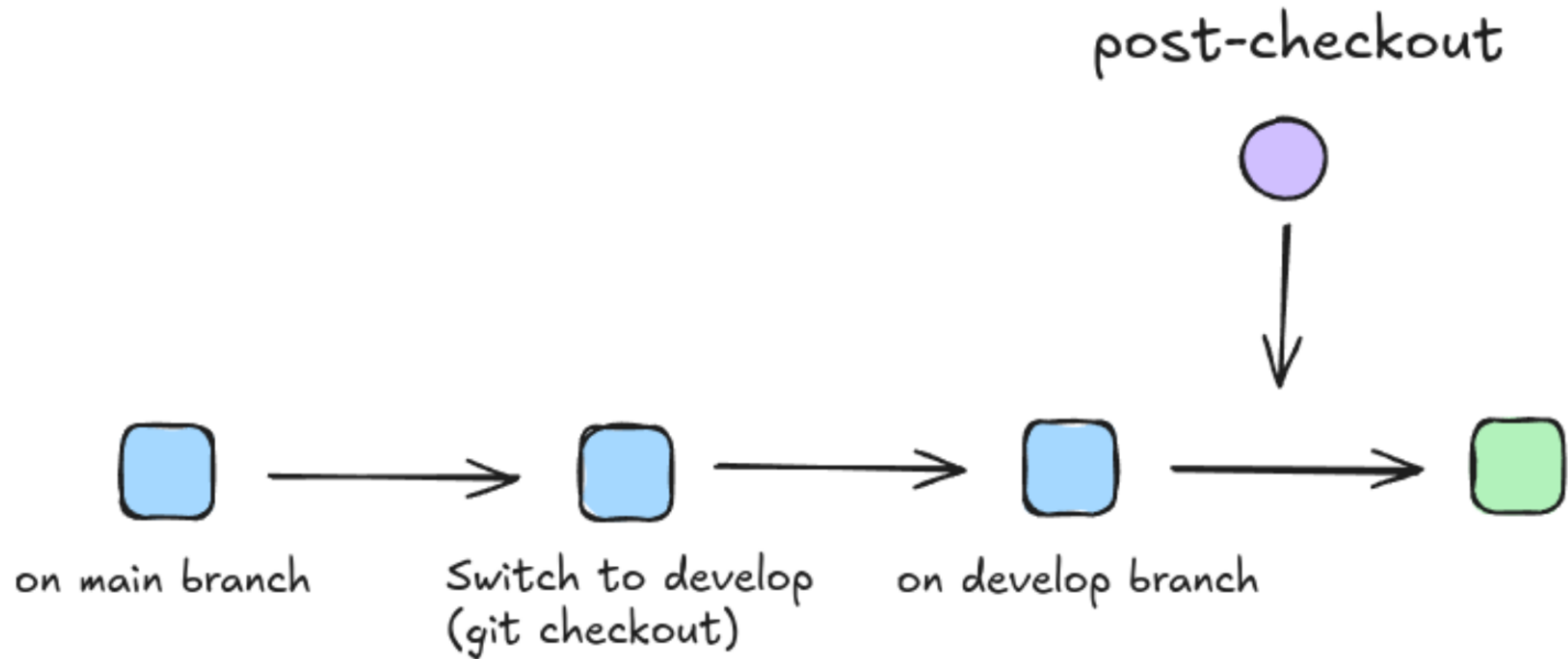
# Agenda

- Conceptual overview
- Common use-cases and tooling
- Practical example: Enforcing code style
- Tips

# git hooks

Scripts that are triggered whenever certain events happen in git repositories.

# Sample: post-checkout



# Requirements

- the name of the script matches the event e.g. pre-commit
- the script is executable
- the script is located where git looks for hooks

# Installation

The default location is in the `.git/hooks` directory in your repository.

A terminal window titled "sample-repo" with three colored window control buttons (red, yellow, green) in the top-left corner. The terminal displays the output of the command `$ tree .git/hooks`. The output shows the directory `.git/hooks/` and a list of sample hook files: `applypatch-msg.sample`, `commit-msg.sample`, `pre-commit.sample`, `prepare-commit-msg.sample`, `pre-push.sample`, `pre-rebase.sample`, `pre-receive.sample`, and `update.sample`. Each file name is preceded by a vertical line representing a tree structure.

```
$ tree .git/hooks
.git/hooks/
├── applypatch-msg.sample
├── commit-msg.sample
├── pre-commit.sample
├── prepare-commit-msg.sample
├── pre-push.sample
├── pre-rebase.sample
├── pre-receive.sample
└── update.sample
```

# Customization

 **Tell git where your hooks are located**

```
git config core.hooksPath /path/to/your/hooks
```



# Pick a scripting language

post-checkout

```
#!/bin/bash
```

```
echo "Hello, World!"
```

post-checkout

```
#!/usr/bin/python3
```

```
print("Hello, World!")
```

post-checkout

```
#!/usr/bin/node
```

```
console.log("Hello, World!")
```



# Client & Server-Side Hooks

## Client-Side

post-checkout

pre-commit

pre-push

## Server-Side

pre-receive

update

post-receive

# Common use-cases

# Automatically install dependencies



## Demo: auto install

post-checkout hook running npm install if node\_modules does not exist

# Formatting of commit messages



## Demo: format commit message

prepare-commit-msg hook including the current branch name in the commit message

# Enforcing code style

Most projects will use *linter* and *formatter*.

*Linters* analyzes and detects common errors in your code.

*Formatters* restructures how code appears.

# Using ruff

```
pipx install ruff  
ruff check /path/to/code --fix --show-fixes # linter  
ruff format /path/to/code # formatter
```

# ruff configuration

```
ruff.toml

[lint]
# Enable Pyflakes (`F`) and a subset of the pycodestyle (`E`) codes by default.
# Unlike Flake8, Ruff doesn't enable pycodestyle warnings (`W`) or
# McCabe complexity (`C901`) by default.
select = ["E4", "E7", "E9", "F"]
ignore = []

# Allow fix for all enabled rules (when `--fix`) is provided.
fixable = ["ALL"]
unfixable = []

# Allow unused variables when underscore-prefixed.
dummy-variable-rgx = "^(_+|(_+[a-zA-Z0-9_]*[a-zA-Z0-9]+?))$"

[format]
# Like Black, use double quotes for strings.
quote-style = "double"

# Like Black, indent with spaces, rather than tabs.
indent-style = "space"

# Like Black, respect magic trailing commas.
skip-magic-trailing-comma = false

# Like Black, automatically detect the appropriate line ending.
line-ending = "auto"

# Enable auto-formatting of code examples in docstrings. Markdown,
# reStructuredText code/literal blocks and doctests are all supported.
#
# This is currently disabled by default, but it is planned for this
# to be opt-out in the future.
docstring-code-format = false

# Set the line length limit used when formatting code snippets in
# docstrings.

# This only has an effect when the `docstring-code-format` setting is
# enabled.
docstring-code-line-length = "dynamic"
```



# Using eslint and prettier

```
npm i --save-dev \  
  eslint \  
  prettier \  
  eslint-config-prettier \  
  eslint-plugin-prettier
```

# eslint and prettier configuration

```
eslint.config.js

import js from "@eslint/js";
import eslintConfigPrettier from "eslint-config-prettier";

export default [
  js.configs.recommended,
  eslintConfigPrettier,
  {
    rules: {
      indent: ["error", 2],
      "linebreak-style": ["error", "unix"],
      quotes: ["error", "double"],
      semi: ["error", "never"],
    },
  },
]
```

```
.prettierrc

{
  "printWidth": 120,
  "semi": false,
  "trailingComma": "all"
}
```

# Tooling

# Rationale

Scripts should be part of the code -- so that changes are tracked and are accessible to everyone in the team.

Tools like pre-commit and husky make it easier to manage and maintain git hooks.

# pre-commit


```
pipx install pre-commit # install pre-commit  
# add a pre-commit configuration .pre-commit-config.yaml  
pre-commit install      # install the git hook scripts
```

# pre-commit configuration

```
.pre-commit-config.yaml

repos:
  - repo: https://github.com/astral-sh/ruff-pre-commit
    rev: v0.8.0
    hooks:
      - id: ruff
        files: src
        args: [ --fix, --show-fixes ]
      - id: ruff-format
```

# Customizing pre-commit

 .pre-commit-config.yaml

```
repos:
  - repo: local
    hooks:
      - id: check-x
        name: Check X
        entry: ./bin/check-x.sh
        language: script
        types: [python]
        stages: [post-checkout]
```

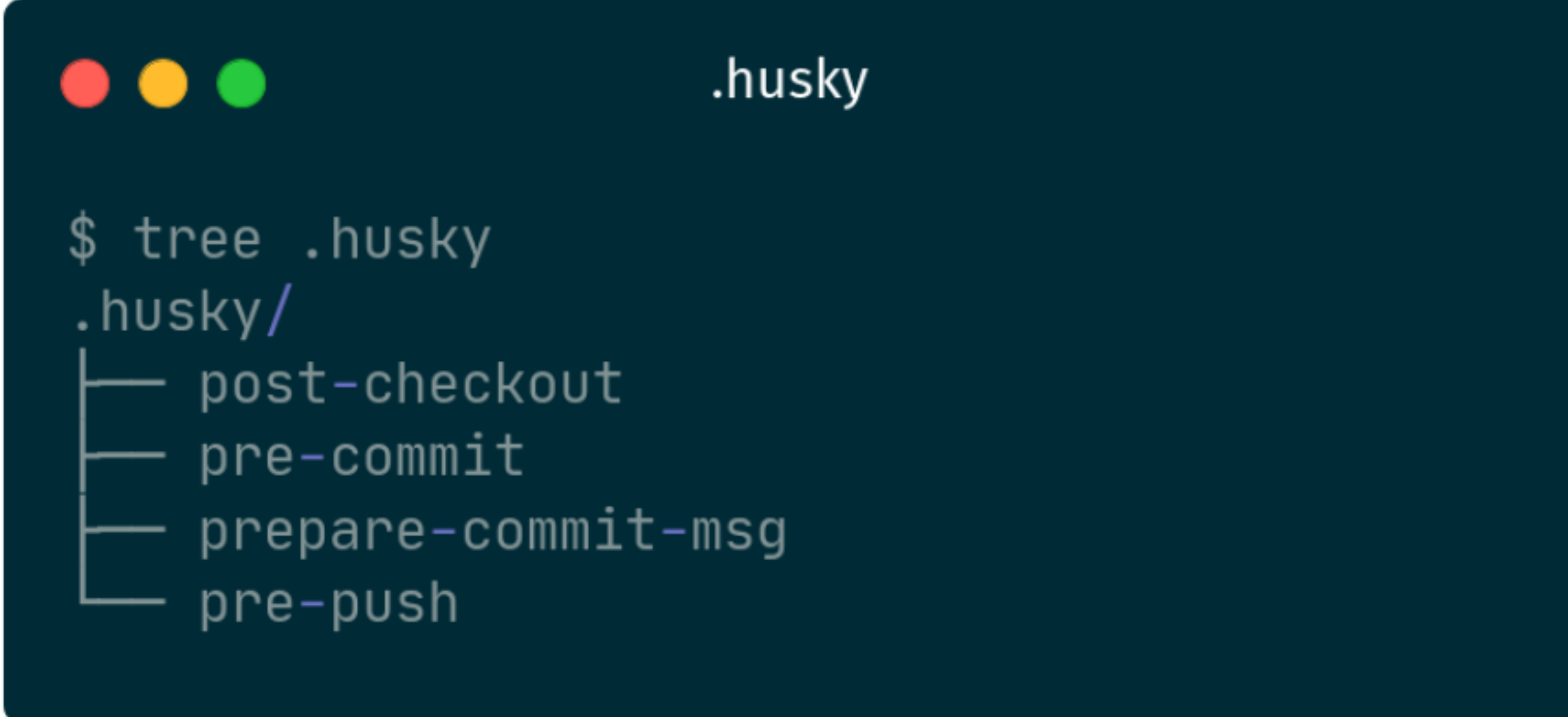


# husky

```
npm install --save-dev husky  
npx husky init      # initializes a .husky/ directory
```

# husky config

husky creates a `.husky` directory where it *installs* your scripts and tell git about it.



```
.husky

$ tree .husky
.husky/
├── post-checkout
├── pre-commit
├── prepare-commit-msg
└── pre-push
```

# lint-staged

```
npm install --save-dev lint-staged  
# after installation add config in either  
# package.json or .lintstagedrc
```

# lint-staged configuration

```
package.json

{
  "lint-staged": {
    "!(*.js)": "prettier --write",
    "*.js": ["eslint --fix", "prettier --write"]
  }
}
```

```
.husky/pre-commit

lint-staged
```

# Tips

- Prefer the long version of command-line flags
- Convention over configuration / sensible defaults
- Tweak the (lint and format) rules as needed
- Experiment and see what works for you and your team

# Repos



pre-commit-ruff-demo



husky-eslint-demo

# Source



[jduabe.dev/git-hooks](https://jduabe.dev/git-hooks)