Coding culture at EEG

Tools and best practices of scientific software development

AG-strategy meeting
Monday, January 17, 2022

Philip Hackstock

Research Software Engineer
Scenario Services Team - Energy, Climate, and Environment Program (ECE)
International Institute for Applied Systems Analysis (IIASA)
Laxenburg, Austria

Please consider the environment before printing this slide deck

Purpose of this workshop

- Provide a starting point for establishing 'coding culture'
  - ⇒ A set of guidelines, tools and processes to make your life better
  - ⇒ Focus on science, not fighting with the tools
  - ⇒ Better and easier collaboration, internal and external
- This is **not** going to be a deep dive into topics
- It should be a reference and starting point for you
- Technical terms to google will be underlined
- Provide useful links to videos and articles
- Teach yourself and teach others

Background about myself

- Graduated with a master's in physics from TU Wien in 2020
  - ⇒ Worked at CERN & MedAustron
  - ⇒ Both related to software
- Started at IIASA in March 2021 as a research software engineer
- Part of the Scenario Services Team
- Mainly work in python

Content of the workshop

- 3x45min with 15 minutes of breaks
- Git
  - ⇒ Basics of version control software
- GitHub
  - ⇒ How to build software together
  - ⇒ Tools & workflows
- Best Practices, common problems & solutions

Acknowledgements

Based on material by:

Daniel Huppmann
(@danielhuppmann)

Paul Natsuo Kishimoto
(@khearu)

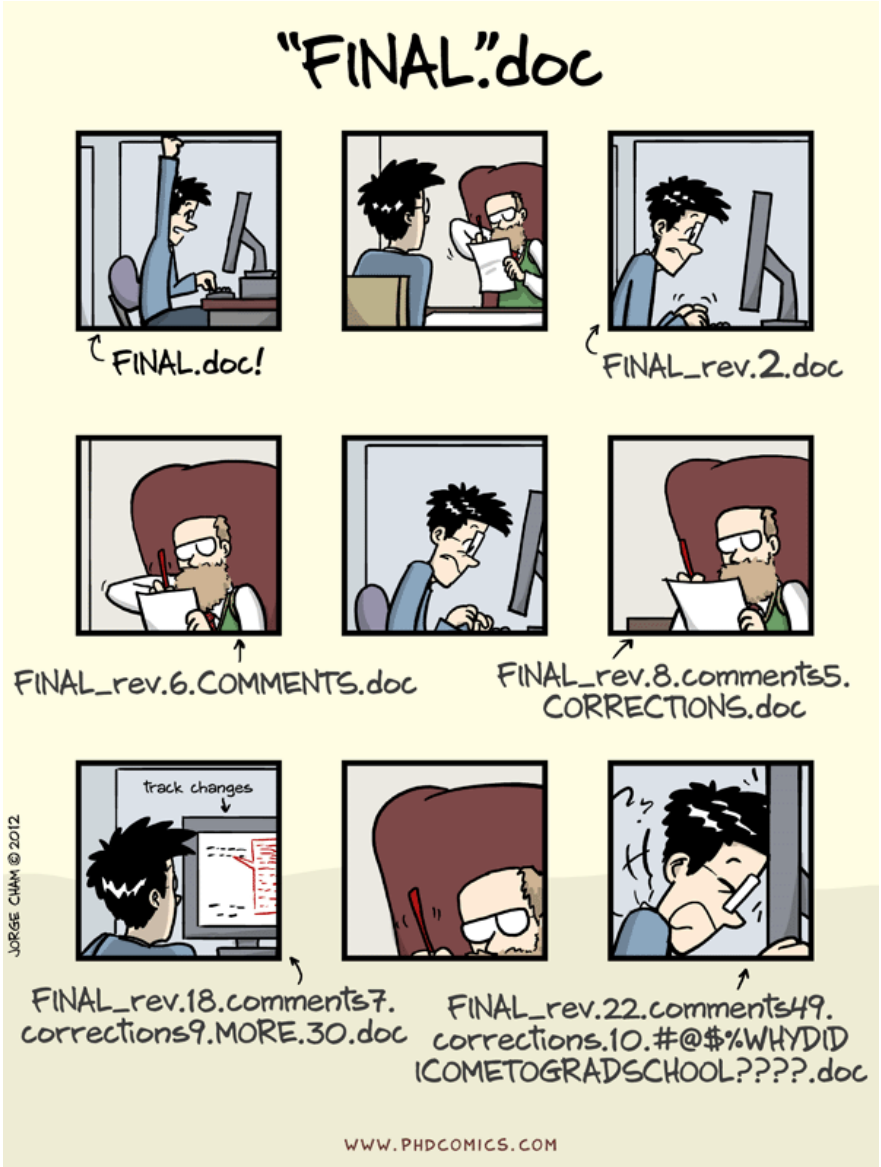
Can be found here: teaching.ece.iiasa.ac.at

Part 1

Git Basics

- What is it and why do we want to use it?
- Basics principles and terms
- Interaction with git
- Commit messages, special files & branches

What is git and why is it useful?



Source: https://phdcomics.com/comics/archive.php?comicid=1531

What is git and why is it useful?

- Git is a (distributed) version control system
- Built for collaboration
- Several hosing providers (GitHub, GitLab) provide additional tools
  - ⇒ User interfaces for code review using pull requests
  - ⇒ Automated tasks
  - ⇒ Issue tracking and discussion, kanban boards, …

A quick introduction to version control using git

- Key differences between git version control vs. folder synchronization (e.g. Dropbox, Google Drive)
  - ⇒ You define the relevant unit or size of a change by making a commit
  - ⇒ Adding comments to your commits allows to attach relevant info to your code changes
  - ⇒ Branches allow you to switch to a "parallel universe" within a version control repository
  - ⇒ It's a decentralized version control tool that supports offline, parallel work
  - ⇒ There is a well-defined routine for merging developments from parallel branches
- Git is great for uncompiled code and text with simple mark-up
  - ⇒ Use other version control tools for data, presentations, compiled software, ...

Before we dive in

How to interact with git

- 2 options
  - ⇒ 'classic' command line
  - ⇒ Graphical user interface e.g. GitKraken, GitHub Desktop, ...
- Use GitKraken for this demonstration
- Get GitKraken pro with academic license
  - ⇒ GitHub Teacher Toolbox
- Need to learn git vocabulary: push, pull, branch, fork, remote, add, commit, etc...

Git demonstration

Nothing has ever gone wrong in a live demo...

So let's do one!

Commit messages

How to style them so they are useful

- Useful for yourself & your collaborators
- Short title (72 characters max)
- Start with a verb in imperative, 'Add', 'Change', 'Fix', etc...
- Use the body for details
- Details can be found here

Special files in git folders

- README
  - ⇒ Explains what the repository contains
  - ⇒ How to use the code
  - ⇒ Displayed by GitHub
- .gitignore
  - ⇒ Defines patterns for files that git will not track
  - ⇒ .gitignore generator https://www.toptal.com/developers/gitignore

Branching models

What is a branch, why should you branch and when?

- Branch is a copy of a repository
- Main branch should always be working
- Don't interfere with colleagues' work
- When developing new features things can (will) break
- Rule of thumb:
  - ⇒ One feature per branch
  - ⇒ Not always feasible but keep it as small as possible
- Two main models: git flow (details) and GitHub flow (details)

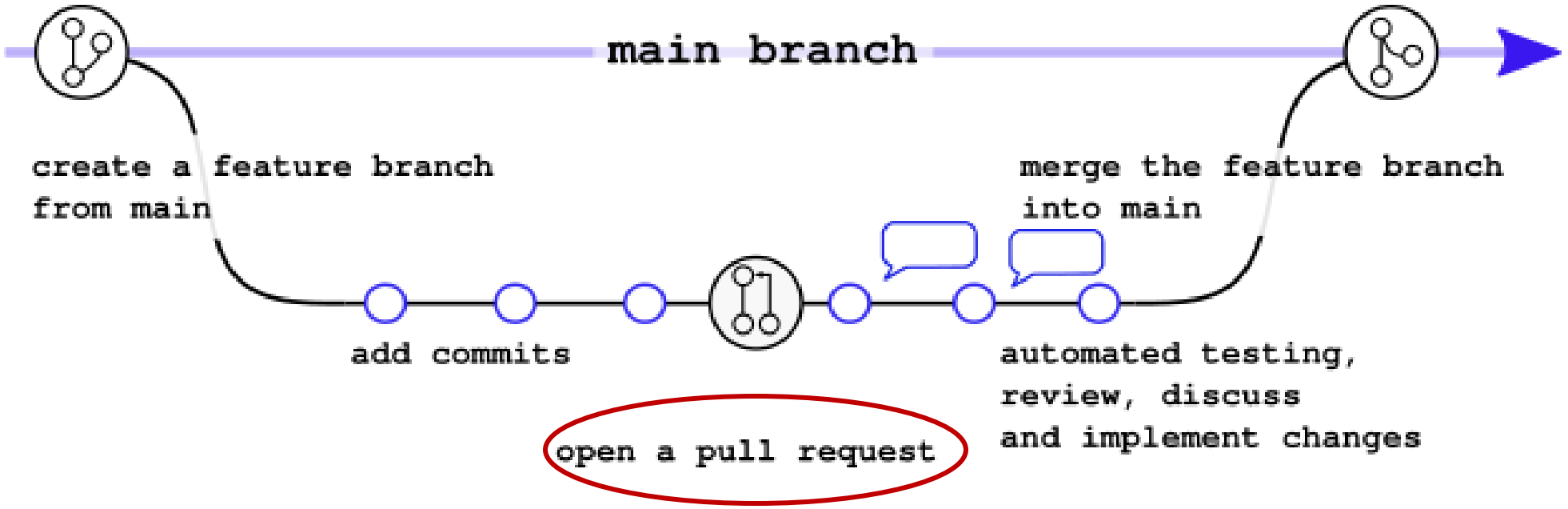
Questions about git

Part 2

GitHub – How to use it

- General overview
- GitHub flow
- CI/CD with GitHub Actions & Unit testing
- Open-source licenses
- GitHub organizations

GitHub flow

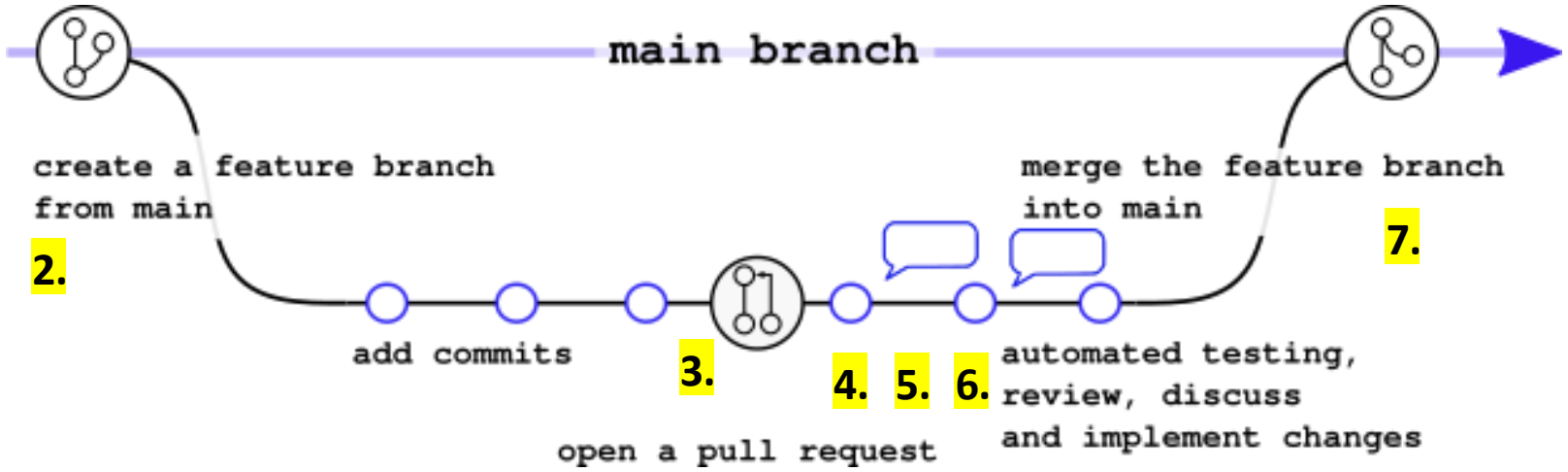


More details on the GitHub documentation: https://docs.github.com/en/get-started/quickstart/hello-world#creating-a-branch

Issue & Pull request workflow

An example timeline

1. Alice finds & creates an issue
2. Alice creates a new local branch and pushes updates
3. Alice opens a PR, self-assigns and chooses Bob as a reviewer
4. Bob requests some changes
5. Alice implements these changes
6. Bob approves
7. Alice merges & deletes the branch



Demonstration on GitHub

How to review a pull request

https://github.com/phackstock/eeg_demonstration

Programming collaboration etiquette

Be kind and respectful in collaboration, code review and comments

- Collaborative scientific programming is about communication, not code...
- Keep in mind that discussions via e-mail, chat, pull requests comments, code review, etc. lack a lot of the social cues that human interaction is built upon
- If there are two roughly equivalent ways to do something and a code reviewer suggests that you use the other approach...
  - ⇒ Just do it their way if there is no good reason not to – out of respect for the reviewer and to avoid getting bogged down in escalating discussions
- Give credit generously to your collaborators and contributors!

This slide was minimally adapted from slide 17 of Open-Source Energy System Modeling, Lecture 2 by Daniel Huppmann (https://data.ene.iiasa.ac.at/huppmann/open-energy-modelling-spring-2021/lecture-2.pdf)

GitHub Actions

Automating repetitive tasks

- Workflow instructions to be run on GitHub's servers
- An implementation of continuous integration **CI** & continuous deployment **CD**
- Setup as event triggered system
- Only free for public repos
- Defined in **yaml** files
- Useful for automated testing, code-style, publishing to pypi, building documentation, ...
- Details in the github action documentation

```yaml
name: Run tests

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2

    - name: Set up Python 3.8
      uses: actions/setup-python@v2
      with:
        python-version: 3.8

    - name: Install dependencies
      run: pip install pytest

    - name: Test with pytest
      run: pytest
```

Source: https://github.com/danielhuppmann/lecture-spring-2021/blob/main/.github/workflows/pytest.yml

Unit testing

- Write pieces of code that test **one** specific aspect
- Run them periodically to make sure you did not introduce any breaking changes
- Automate them using GitHub Actions to ensure that only code that passes tests is pushed
- Different frameworks to choose from (unittest, **pytest**, etc...)
- Check code coverage -> no guarantee for correct code though

Licenses

Why do we need them and where do we start?

- Per default creative work is copyrighted
- We need to attach an open-source license to allow people to use it
- Two main types
  - ⇒ *Permissive:* people are free to do whatever
  - ⇒ *Copy left:* All modifications must be redistributed under the same open license
- choosealicense.com is a helpful resource to find the right one
- Add a file called LICENSE to your GH repo

GitHub organization

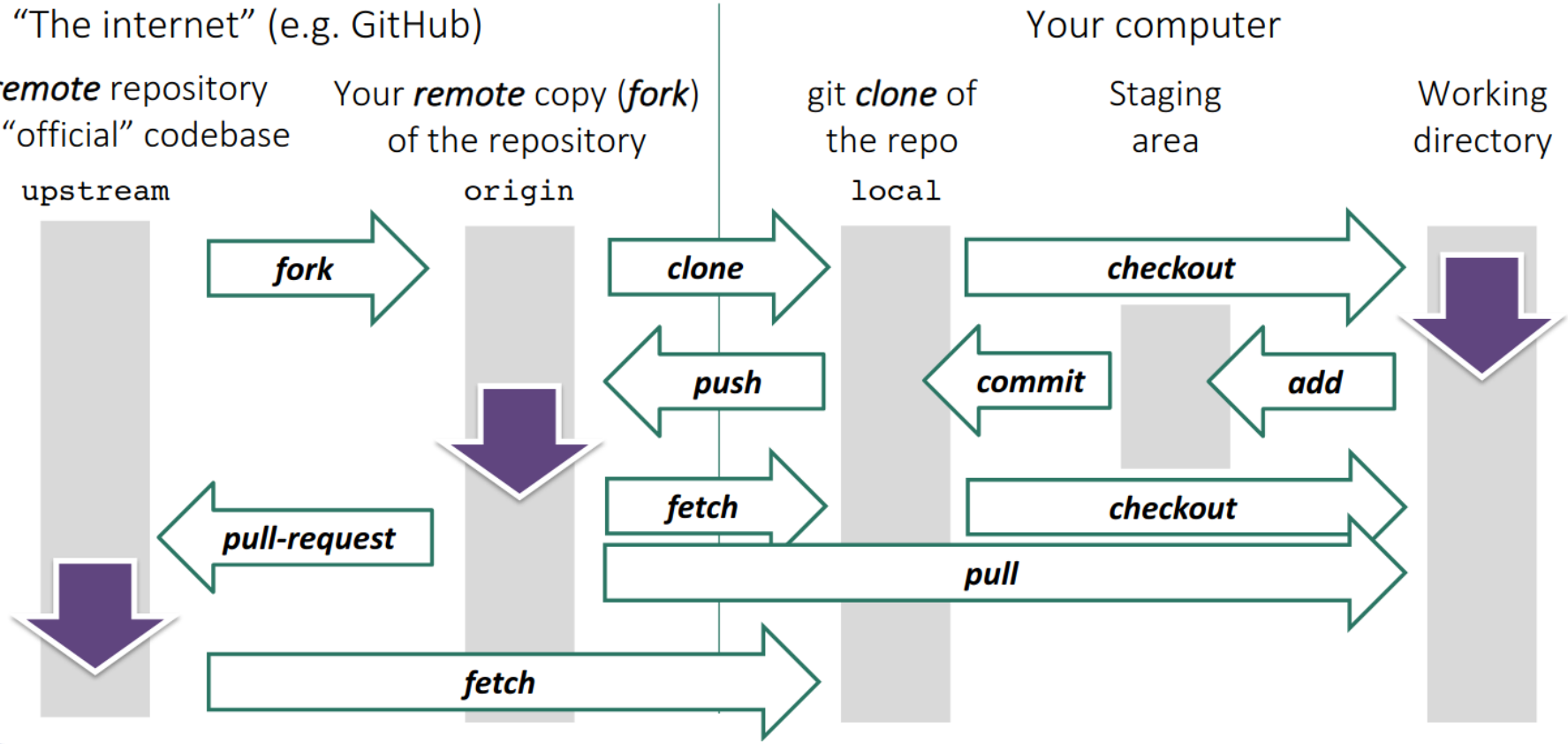
A useful way to collect all your code

- Code is hosted under a GitHub organization
- Teams can be created to easily collaborate

  ⇒ Different levels of permissions (read, write, admin)
- GitHub actions & other premium features are only free on public repos
- An organization can pay so that GitHub Actions are available for private repositories

Examples of bigger projects

- **Fork** & pull request workflow
- Examples: pyam, nomenclature



Questions about GitHub

Part 3

Best practices

- Type hints & docstrings
- Formatters & linters
- Packaging

Type hints & docstrings

- Python now supports type hints (example from nomenclature)
- Purely informational, does not affect the program
- Mypy does type checking
- Integrates with **vs code** and other IDEs
- **Docstrings** for functions & classes
  - ⇒ Automated generation possible
  - ⇒ Different styles (details)
- Turn doc strings into full documentation with sphinx & host on readthedocs
- pyam and nomenclature example

Formatting & linting

Better code-style for better programs

- Formatting: making sure the code adheres to standards (PEP8)
- Linting: checking for syntax and style problems
- Number of tools for both (black, etc...)
  - ⇒ For formatting: black
  - ⇒ For linting: Flake8
- Can be integrated into an IDE
- Can be integrated into GitHub using stickler
  - ⇒ Only free for public repos

Packaging

How to make software installable

- Pseudo-installable with requirements.txt
- Use of a virtual environment and pip freeze
- Python packaging is a bit of a mess
- Different ways to do it with: setup.py, setup.cfg, pyproject.toml

  ⇒ Example: nomenclature
- Using poetry is an easy solution
- Publish your package to pypi so that it can be 'pip installed'

Resources

Further information

- Git & GitHub:
  - ⇒ Official GitHub documentation
  - ⇒ Two example repos, eeg_demonstration, eeg_live_demonstration
- Scientific software development best practices:
  - ⇒ teaching.ece.iiasa.ac.at, Daniel Huppmann's and Paul Natsuo Kishimoto's lectures
- Youtube channels:
  - ⇒ ArjanCodes, mCoding
- Books:
  - ⇒ High-performance python

Thank you very much for your attention

Thanks to Daniel Huppmann (@danielhuppmann) and Paul Natsuo Kishimoto (@khaeru) for sharing their teaching material

Philip Hackstock

Research Software Engineer

Scenario Services Team - Energy, Climate, and Environment Program (ECE)
International Institute for Applied Systems Analysis (IIASA)
Laxenburg, Austria

Please consider the environment before printing this slide deck