International Institute for Applied Systems Analysis
IIASA www.iiasa.ac.at

TU WIEN

Lectures 1 & 2: Principles of open-source and collaborative scientific programming for energy modelling

Open-Source Energy System Modeling
TU Wien, VU 370.062

Dipl.-Ing. Dr. Daniel Huppmann

Please consider the environment before printing this slide deck

Required reading and preparation

- Preparation for scientific programming exercises in this lecture:
  - ⇒ create a GitHub account
  - ⇒ know what it means to 'clone' a repository, make a 'commit' and 'push'
  - ⇒ either get familiar with 'git' using the command line
  - ⇒ or get familiar with a program for working with git repos (for novice users, try Gitkraken)
  - ⇒ Install Python (for novice users, try Anaconda)
  - ⇒ get familiar with the basic Python syntax

- Required reading:
  - ⇒ The FAIR Guiding Principles, Mark Wilkinson et al. *Scientific Data* 3:160018 (2016)
    doi: 10.1038/sdata.2016.18
  - ⇒ Greg Wilson et al. Good enough practices in scientific computing.
    PLOS Computational Biology 13(6), 2017. doi: 10.1371/journal.pcbi.1005510

Background: Climate change mitigation and energy system transformation

Following the approval of the IPCC Special Report on Global Warming of 1.5°C, media & newspapers widely quoted required system transformations

Harry Taylor, 6, played with the bones of dead livestock in Australia, which has faced severe drought.
Brook Mitchell/Getty Images

The New York Times

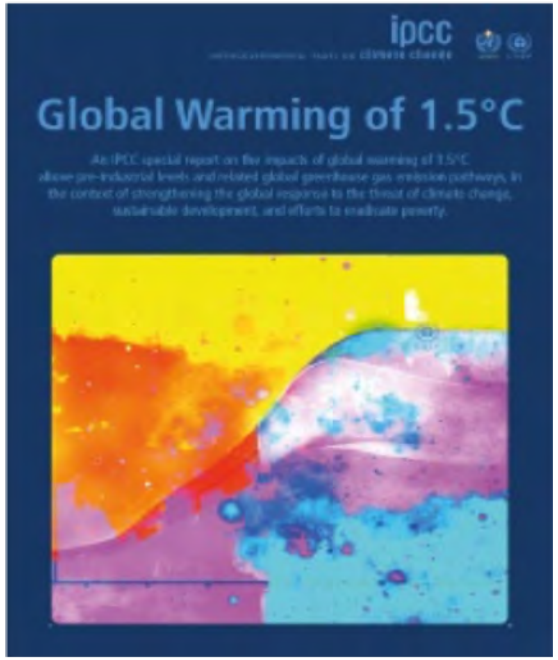
Major Climate Report Describes a Strong Risk of Crisis as Early as 2040

[…] To prevent 2.7 degrees of warming, the report said, greenhouse pollution must be reduced by 45 percent from 2010 levels by 2030, and 100 percent by 2050. It also found that, by 2050, use of coal as an electricity source would have to drop from nearly 40 percent today to between 1 and 7 percent. Renewable energy such as wind and solar, which make up about 20 percent of the electricity mix today, would have to increase to as much as 67 percent. […]

www.nytimes.com/2018/10/07/climate/ipcc-climate-report-2040.html

Where do these numbers come from?

The IPCC *Special Report on Global Warming of 1.5°C* (SR15) was published in the fall 2018.
www.ipcc.ch/sr15



Overview of the lecture

We will dive into the assessment of energy system transformation pathways while discussing the key concepts of collaborative scientific programming

Content and teaching goals:

- Introduction to scientific programming and open-source software/data (**Lecture 1**)
  ⇒ What is it, why do we it, how do we do it?
- Integrated assessment of climate change & sustainable development (**Lectures 2 & 3**)
  ⇒ How can scenarios from these models be used in scientific assessment like the IPCC SR15?
  Using *Jupyter* notebooks and the *pyam* package for scenario analysis
  (software.ene.iiasa.ac.at/pyam)
- Development of a national energy system model for policy evaluation (**Lectures 4 & 5**)
  ⇒ How can we develop scenarios to analyse climate policy measures?
  Using the open-source MESSAGE$_{ix}$ energy modelling framework (MESSAGEix.iiasa.ac.at)

Course structure and lecture content subject to change depending on feedback and interest!

About myself: education and research career

From mathematics to energy economics and climate policy

- Dipl.-Ing. (MSc) in Mathematics at TU Wien, specialization *Mathematics in Economics*
- Researcher at the "German Institute for Economic Research" (DIW Berlin)
- Doctorate at TU Berlin in Operations Research, Game Theory and Energy Economics
- Postdoctoral Fellowship at Johns Hopkins University, Baltimore
- Research Fellow at "Resources for the Future" (think-tank in Washington D.C.)
- Research Scholar (since October 2015) at the Energy Program, International Institute for Applied Systems Analysis, Laxenburg
- Contributing Author and Chapter Scientist of the IPCC's *Special Report on Global Warming of 1.5°C* (SR15) published in October 2018

Overview of the lecture (II)

The correct use of collaborative tools and workflows will be as important as the application to a problem and correct interpretation of the results

Requirements:

⇒ A good understanding of energy systems and climate policy

⇒ Experience with at least one scientific programming language

Mode of exercises:

⇒ Submit assignments via GitHub pull requests and Scenario Explorer workspaces

Grade:

⇒ Submitted assignments (50%)

⇒ Oral discussion of submitted exercises and related questions (30%)

⇒ Active participation in class – feel free to ask questions any time (20%)

About you...

What is your background and experience level with (scientific) programming?

Excel
python
R
C++
Java
julia
MATLAB
GitLab
GitHub
LaTeX
GAMS
AMPL

Microsoft Excel as a programming language?

People tend to have strong feelings about Excel...

John Oliver, *Last Week Tonight,* June 5, 2016. Meme from memegenerator.net, Clip on youtube.com

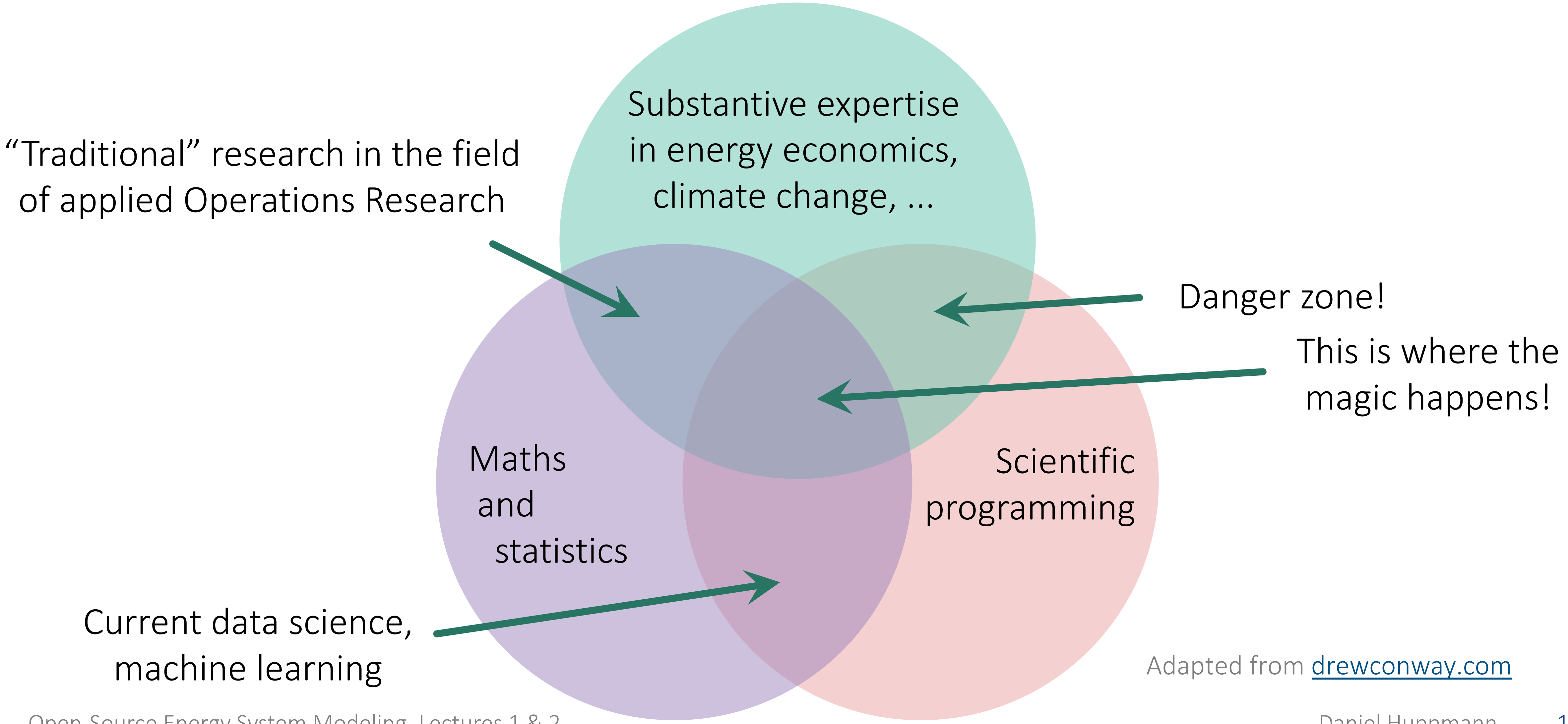
Part 1

An introduction to open, collaborative scientific research

Based on material by Matthew Gidden (@gidden) and Paul Natsuo Kishimoto (@khaeru)

The intersection between energy economics and mathematics

Current research requires substantial expertise in scientific programming



Adapted from drewconway.com

Key misconceptions about best practice in open scientific programming

If you think that this topic is of no concern to you, you're probably wrong

- Who is your main (and usually worst) collaborator?

  ⇒ Yourself from six months ago!
  Because you did not write enough documentation and don't respond to emails anymore

- Why is it a bad idea to use data or software that does not have an open license?

  ⇒ Bad karma!

  ⇒ Are you intending to distribute your work?
  How are you planning to deal with the parts that your project depends on?

- Why should you share data and code under an open-source license?

  ⇒ Good karma!

  ⇒ Standard licenses have a disclaimer of liability, so you cannot be accountable for problems

  ⇒ There is probably a growing expectation from your (potential) collaborators

  ⇒ *Treat your GitHub, etc. profile as your "business card" similar to your list of publications*

Licensing – free and/or open-source software

Freedom in science is not about the price – it's about what you're allowed to do

- Per default, a creative work including software code attracts copyright
  ⇒ The authors (or the employer) retains all rights on how the work may be used be others
- *Free software* is not quite the same as *open-source*
  → defined by "Four Freedoms" →
  ⇒ In practice, the terms are used interchangeably
- Two classes of free/open software licenses distinguished by limitations on redistribution:
  ⇒ *Permissive*: No restrictions on redistribution, including the right *not to share* derivative work
  ⇒ *Copyleft*: All modifications must be redistributed under the same open license

Freedom 0: To ***run*** the program for any purpose.
Freedom 1: To ***study*** how the program works, and change it to make it do what you wish.
Freedom 2: To ***redistribute*** and make copies so you can help your neighbour.
Freedom 3: To ***improve*** the program, and release your improvements/modifications to the public.

The first formal definition of free software was written by Richard Stallmann for the *Free Software Foundation*. GNU's Bulletin 1(1):8, February 1986. Via Wikipedia.

To find out which license is appropriate for your project: choosealicense.com

The FAIR Guiding Principles

Existing digital ecosystem of scholarly data publication prevents us from extracting maximum benefit from our research investments

- Good data management and stewardship is not a goal in itself
  ⇒ Rather, it's a pre-condition supporting knowledge discovery and innovation.
- Increasingly, science funders, publishers and governmental agencies require data management and stewardship plans publicly funded research projects
- Digital research objects should be available for **transparency**, **reproducibility** and **reusability**
  ⇒ This includes data as well as algorithms, tools and workflows to compile and assess data
- Data management must be geared towards human readers *and* machine processing
  ⇒ Humans have an intuitive sense of 'semantics' (the meaning or intent of a digital object)
  ⇒ But humans are not able to operate at the scope, scale, and speed required for the scale of contemporary scientific data and complexity

Mark Wilkinson et al. *Scientific Data* 3:160018 (2016) doi: 10.1038/sdata.2016.18

The distinction between FAIR for humans vs. machines

Humans are good at parsing information...

International Institute for Applied Systems Analysis
IIASA www.iiasa.ac.at

IAMC

IAMC 1.5°C Scenario Explorer and Data hosted by IIASA

Daniel Huppmann, Elmar Kriegler, Volker Krey, Keywan Riahi, Joeri Rogelj, Steven K. Rose, John Weyant, Nico Bauer, Christoph Bertram, Valentina Bosetti, Katherine Calvin, Jonathan Doelman, Laurent Drouet, Johannes Emmerling, Stefan Frank, Shinichiro Fujimori, David Gernaat, Arnulf Grubler, Celine Guivarch, Martin Haigh, Christian Holz, Gokul Iyer, Etsushi Kato, Kimon Keramidas, Alban Kitous, Florian Leblanc, Jing-Yu Liu, Konstantin Löffler, Gunnar Luderer, Adriana Marcucci, David McCollum, Silvana Mima, Alexander Popp, Ronald D. Sands, Fuminori Sano, Jessica Strefler, Junichi Tsutsui, Detlef Van Vuuren, Zoi Vrontisi, Marshall Wise, Runsen Zhang. **IAMC 1.5°C Scenario Explorer and Data hosted by IIASA**. *International Institute for Applied Systems Analysis & Integrated Assessment Modeling Consortium*. (2018) 10.22022/SR15/08-2018.15429

Item Type: Dataset

Archive
iamc15_scenario_data_all_regions_r1.1.xlsx - Published Version (Release 1.1)
iamc15_scenario_data_all_regions_r1.xlsx - Published Version (Release 1.0)

Please access this resource from the **Scenario Explorer Website**. For usage rights please see our license here.

Version History:

Release 1.1 (February 7, 2019)
This release includes additional timeseries data to increase reproducibility of the figures and tables in the SR15, and it corrects a number of data issues identified since Release 1.0. None of the changes have any impact on the assessment in the SR15.

Release 1.0 (October 15, 2018)
Scenario ensemble release for the soft launch of the IPCC SR15 following the approval plenary in Incheon, Republic of Korea.

Please view the About page for details.

Rendered version of the landing page for doi 10.22022/SR15/08-2018.15429

The distinction between FAIR for humans vs. machines

Computers require additional structure to parse information

```
<h1>IAMC 1.5°C Scenario Explorer and Data hosted by IIASA</h1>
<Creator:Author>Daniel Huppmann</Creator:Author>
...
<CreationName:Title><b>IAMC 1.5°C Scenario Explorer and Data
hosted by IIASA</b></CreationName:Title>
<Agent:Publisher><i>International Institute for Applied Systems Analysis
& Integrated Assessment Modeling Consortium</i></Agent:Publisher>.
<DateOfPublishing>(2018)</DateOfPublishing>
<Name:Identifier:DoiName>
<a target="_blank" href="https://doi.org/10.22022/SR15/08-2018.15429">
10.22022/SR15/08-2018.15429</a></Name:Identifier:DoiName>
Item Type: <Type>Dataset</Type>
Please access this resource from the <Digital:Website><a target="_blank”
href="https://data.ene.iiasa.ac.at/ iamc-1.5c-explorer">
<b>Scenario Explorer Website</b></a></Digital:Website>.
```

Source code of landing page for doi 10.22022/SR15/08-2018.15429

The FAIR Guiding Principles (II)

Scientific work should be Findable, Accessible, Interoperable and Reusable

Data and/or metadata...

Findable	• F1. ... are assigned a unique and persistent identifier (Digital Object Identifier, DOI) • F2. ... are described with rich metadata (defined by R1 below) • F3. ... clearly and explicitly include the identifier of the data it describes • F4. ... are registered or indexed in a searchable resource (including Google)
Accessible	• A1. ... are retrievable by their identifier using a standardized protocol • A2. ... are accessible, even when the data are no longer available
Interoperable	• I1. ... use a formal, shared, applicable language for knowledge representation • I2. ... use vocabularies that follow FAIR principles • I3. ... include qualified references to other (meta)data
Reusable	• R1. ... are richly described with a plurality of accurate and relevant attributes: clear data license, detailed provenance, meet community standards

Adapted from Box 2: The FAIR Guiding Principles, Mark Wilkinson et al. *Scientific Data* 3:160018 (2016) doi: 10.1038/sdata.2016.18

More common misconceptions about open scientific programming

There are many arguments against open-source – almost none are valid

- "I put all my source code/data on my website, so it is open!"
  - ⇒ This is only true if you added an approved open-source license
  - ⇒ Otherwise, don't use the term *open*, because it can be (mis)understood as *free software*
- "My code/data is open because I'll just send a copy to anyone who asks"
  - ⇒ This is not *open* or *free* according to the common understanding in the community
- "If I make release my code/data under an open-source license, some people may misuse it!"
  - ⇒ If you don't make it openly available, nobody is going to use it at all
- "My code/data can't have a DOI because there are proprietary data included..."
  - ⇒ The DOI is only attached to the metadata of the object, so there is no problem
- "I can't release my code/data now because I have to clean it first and write documentation"
  - ⇒ If that is your approach to scientific programming, you're doing it wrong...

Reproducibility is key to good scientific research

Some examples of what's reproducible… not!

Archiving

Definition: Permanent, incorruptible (as far as possible) storage of code, data or results

⇒ Data or results can be preserved, yet may be impossible to recreate (or just understand).

Version control

Definition: VC tracks changes to software source code or data over time.

⇒ VC can be used by one person and yet be unintelligible (i.e., not reproducible) to another.

Testing & quality control

Definition: Implementation of checks to verify that software and data behave as expected.

⇒ Reproducibility of the analysis for one research project doesn't prevent the next researcher from 'breaking' (de-calibrating, misusing) a model or piece of software.

Recommended further reading:

Barnes (2010). Publish your computer code: it is good enough. *Nature* 467(753):775. doi: [10.1038/467753a](10.1038/467753a)

Barba (2016). The hard road to reproducibility. *Science* 354(6308):142. doi: 10.1126/science.354.6308.142

The rationale for proper version control tools

In love and in scientific research, there is no such thing as "final"…

Adapted from "notFinal.doc" at "Piled Higher and Deeper" by Jorge Cham, http://phdcomics.com

A quick introduction to version control using `git`

Git is so much more than just keeping track of code changes over time

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

Several `git` implementations (e.g., GitHub) provide additional project management tools

- ⇒ User interfaces for code review using ***pull requests***
- ⇒ Issue tracking and discussion, kanban boards, ...

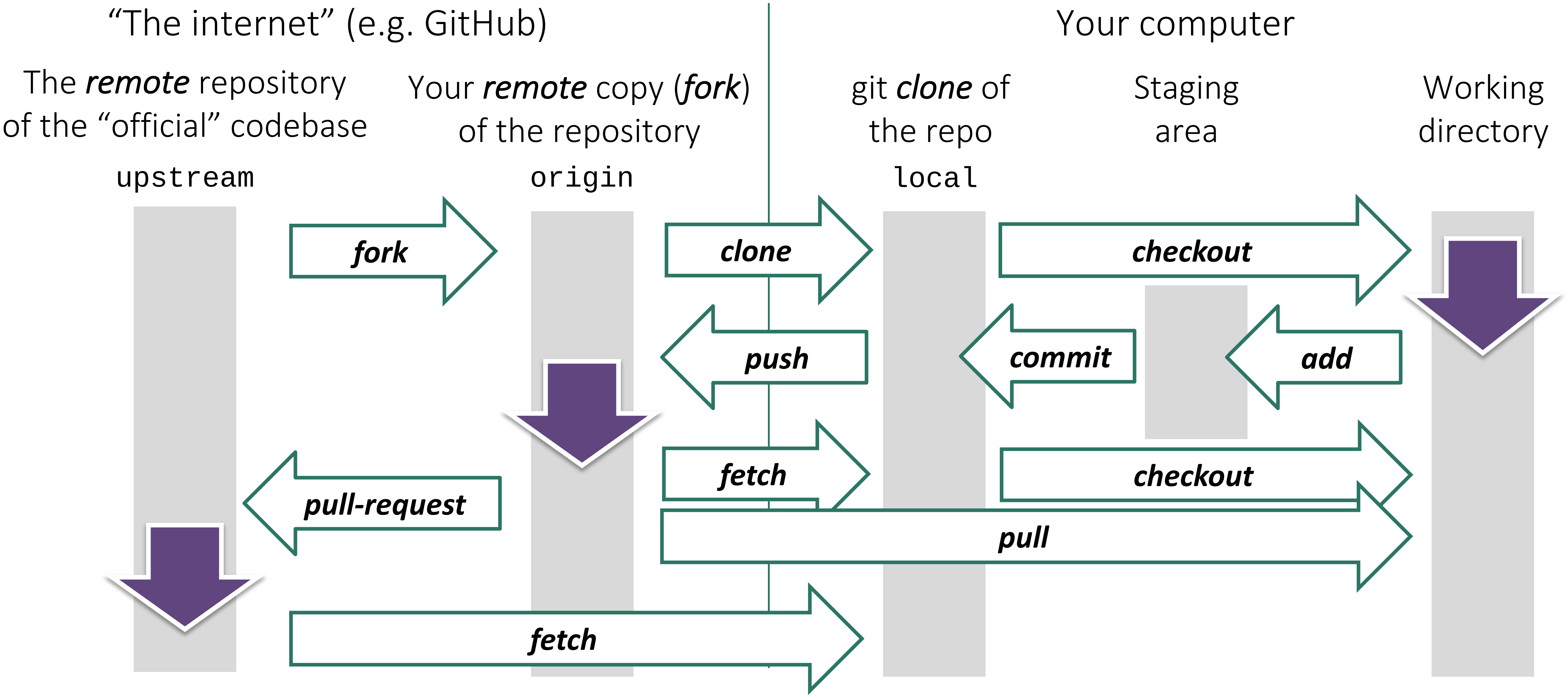
GitHub

However, keep in mind that `git` is great for uncompiled code and text with simple mark-up

- ⇒ Use other version control tools for data, presentations, compiled software, ...

A full `git` workflow

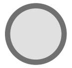
Git is a decentralized version control system geared for collaboration

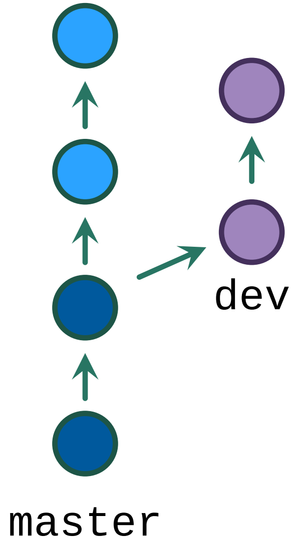


Branching and merging with `git`

There are multiple methods to bring parallel developments back together

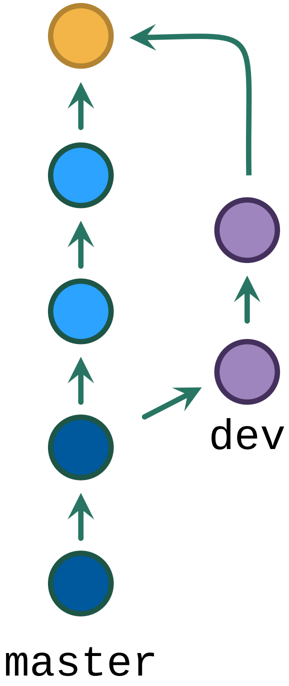
Getting started with branching


... a commit

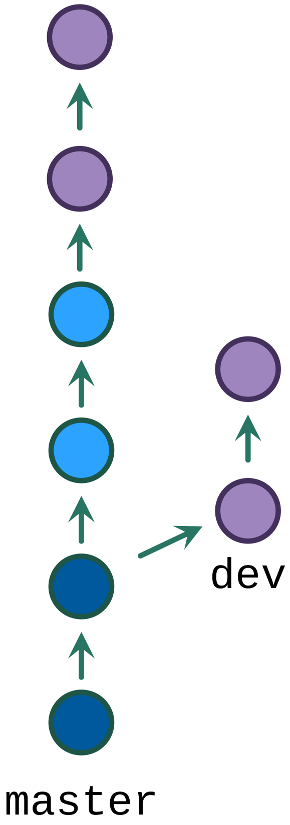


Three options to ***merge*** the changes from `dev` into `master`

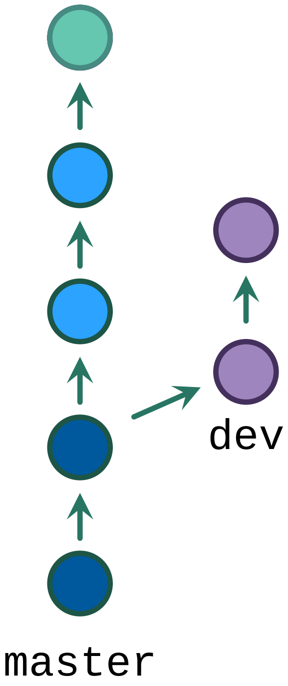
1) A merge commit



2) Rebase



3) Squash and merge



Part 2

Setting up a simple repository with unit tests and continuous integration

The first rule of live demos: Never do a live demo. So let’s do a live demo.

Hands-on exercise

- Set up a new public GitHub repository at www.github.com
- Update the README (formatting using markdown)
- “Clone” the repository to your computer (recommended for novices: gitkraken.com)
- Add a license (why not start with APACHE 2.0?)

  ⇒ Add the statement and the badge to the readme
- Start developing a little Python function (recommended for novices: anaconda.com)
- Add a unit test
- Add a gitignore file
- Add continuous integration using a new branch

  ⇒ travis-ci.com to execute unit tests

  ⇒ stickler-ci to implement linter and code style verification
- Create a pull request to execute the CI and merge the new branch into master
- Add contributing guidelines, set up templates for pull requests
- Create a release

Hands-on exercise (Part II)

- https://github.com/danielhuppmann/lecture_live_demo_2019

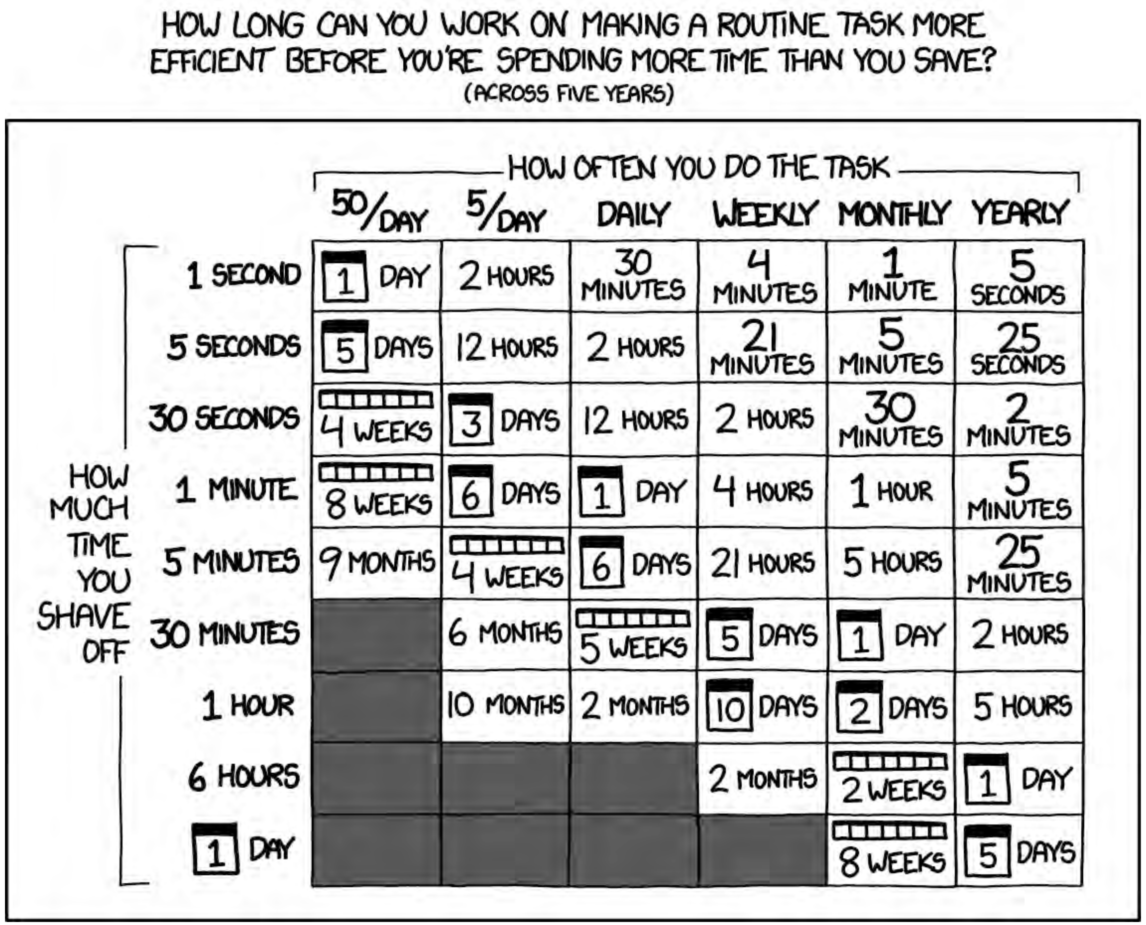
- If a non-admin user wants to push commits, you have to "fork" the repo (create a copy under your GitHub user)
- Clone the fork to your computer
- Start a new branch
- Add a new function or extend some feature such that the unit tests fail
- Make a pull request to the upstream repository
- Fix the code such that unit tests pass
- Ask someone else to perform code review
- Merge the new development (by an admin)

Part 3

Some practical considerations and advice

Time allocation for increasing efficiency through automation

Is it worth the time to automate repetitive tasks? Probably not really…



xkcd by Randall Munroe

Good enough scientific programming

You don’t have to have a PhD in IT to do decent scientific programming!
In fact, it might actually help...

Data management:

⇒ save both raw and intermediate forms, create tidy data amenable to analysis

Software:

⇒ write, organize, and sharing scripts and programs used in the analysis following best practices

Collaboration:

⇒ make it easy for existing and new collaborators to understand and contribute to a project

Project organization:

⇒ organize the digital artefacts of a project to ease discovery and understanding

Manuscripts:

⇒ write manuscripts with a clear audit trail and minimize manual merging of conflicts

Adapted from Greg Wilson et al. Good enough practices in scientific computing. *PLoS Comput. Biol.* 13(6), 2017.
doi: 10.1371/journal.pcbi.1005510

Good enough scientific programming – Software

Your worst collaborator? Yourself from six months ago...

- Place a brief explanatory comment at the start of every program.
- Do not comment and uncomment sections of code to control a program's behaviour.
- Decompose programs into functions, and try to keep each function short enough for one screen.
- Be ruthless about eliminating duplication.
- Always search for well-maintained software libraries that do what you need.
- Test libraries before relying on them.
- Give functions and variables meaningful names.
- Make dependencies and requirements explicit.
- Provide a simple example or test data set.
- Submit code to a reputable DOI-issuing repository (e.g., zenodo).

Adapted from Greg Wilson et al. Good enough practices in scientific computing. *PLoS Comput. Biol.* 13(6), 2017.
doi: 10.1371/journal.pcbi.1005510

Code style guides

Programming should be seen as a (not foreign) language

Which programming language to use, which other conventions to follow?

⇒ If you don't have a strong preference: follow the community or your room (office) mate!

Some practical guidelines:

⇒ Follow a suitable coding etiquette, e.g., PEP8 for Python, Google's R style guide

⇒ For larger projects, agree on a folder structure and hierarchy early (source data, etc.)

⇒ Only change folder structure when it's really necessary

⇒ For more complex code (e.g., packages), use tools to automatically build documentation such as Sphinx and readthedocs.org

Read *the* Docs

Keep in mind...

⇒ Code is read more often than it is written

⇒ Good code should not need a lot of documentation

⇒ Key criteria: readability and consistency with (future) collaborators *and yourself!*

Software releases and semantic versioning

If a piece of software is used by multiple people, clear versioning is critical

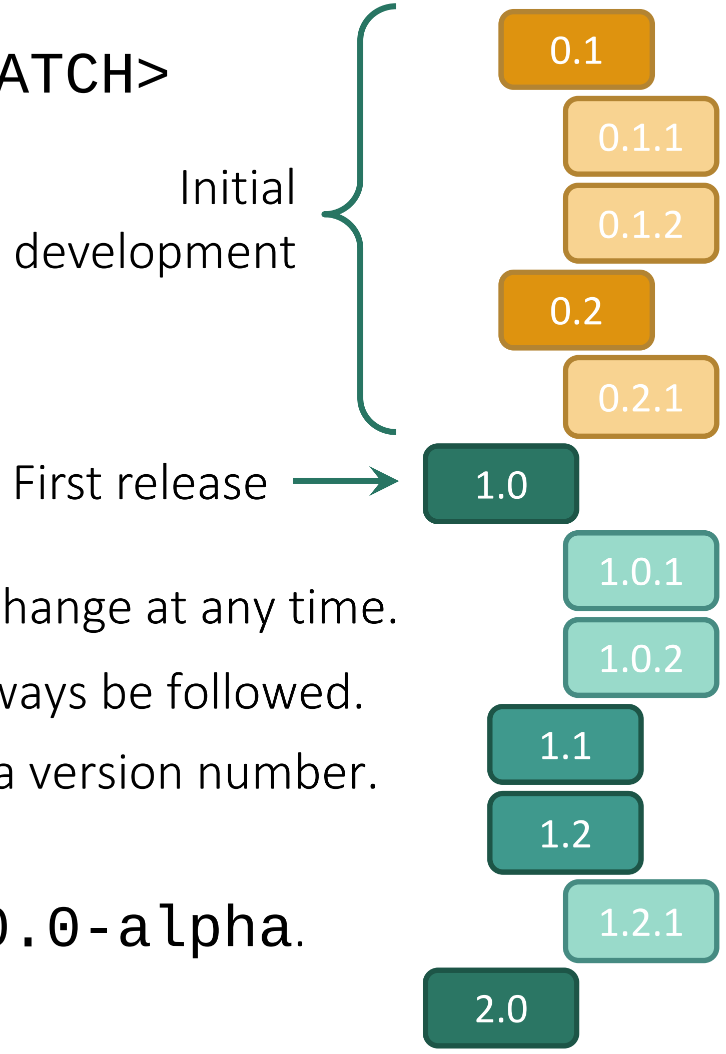
Semantic versioning uses a structure like `<MAJOR>.<MINOR>.<PATCH>`

For a new release (i.e., a published version), you *MUST* increment...

⇒ `MAJOR` when making incompatible API changes,

⇒ `MINOR` when adding backwards-compatible functionality,

⇒ `PATCH` when making backwards-compatible bug fixes.



Other considerations:

- Major version zero (`0.y.z`) is for initial development. Anything may change at any time.
- Version `1.0.0` defines the public API. After that, rules above must always be followed.
- Downstream version numbers *MUST* be reset to 0 when incrementing a version number.
- You *MAY* increment when substantial new internal features are added.
- A pre-release version *MAY* be denoted by appending a string, e.g., `1.0.0-alpha`.

Adapted from Semantic Versioning 2.0.0, semver.org

Coding etiquette

Keep in mind that the internet remembers everything

When you search for my colleague Matthew Gidden on Twitter, the first tweet you find is...

Social etiquette

Be kind and respectful in collaboration, code review and comments

Collaborative scientific programming is about communication, not code

⇒ It's the people, stupid!

⇒ And don't be annoyed when, sometimes, some collaborators are stupid...

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 her/his 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!

International Institute for Applied Systems Analysis
IIASA www.iiasa.ac.at

TU WIEN

Thank you very much for your attention!

Many thanks to Matthew Gidden (@gidden) and Paul Natsuo Kishimoto (@khaeru)
for sharing their lecture material and experience with collaborative programming

Dr. Daniel Huppmann

Research Scholar – Energy Program

International Institute for Applied Systems Analysis (IIASA)
Schlossplatz 1, A-2361 Laxenburg, Austria

huppmann@iiasa.ac.at
http://www.iiasa.ac.at/staff/huppmann