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

Collaborative development of academic software

Paul Natsuo Kishimoto
<paul.kishimoto@iiasa.ac.at>

MESSAGE workshop — Thursday, 10 June 2021

IIASA Energy, Climate, and Environment program

Colophon

These slides have been used for:

- MESSAGEix training & capacity-building workshops given by the IIASA Energy Program on 2020-06-12, 2020-09-10, and 2021-06-10.
- A tutorial on 2021-02-05 with participants from the International Transport Forum at the OECD and IIASA/ECE.

A different subset of the slides was used on each occasion.

LaTeX source, copyright, & license available at `https://github.com/khaeru/doc/`.

Syllabus

C. MESSAGE-GLOBIOM research.
   1. **Validity & reproducibility (r13y) for modeling & scenario research.** Internal vs. external validity. Perspectives on models. Validity vs. r13y. Modeling practices for r13y.

D. MESSAGEix development.
   2. **Version control using `git` & GitHub.** Version control in general. `git` and concepts: commit, branch, diff, tag, repo, merge, fetch/pull/push.
   3. **Collaborative development using GitHub.** Collaboration decisions. GitHub flow. Issue, pull request, milestone, project. Examples & demo.
   4. **Test-driven development.** Purpose. Types of tests. Test coverage. `pytest` intro.
   5. **Continuous integration.** Purpose. GitHub Actions, RTD, Codecov, Stickler.

Outline

C1. Reproducible research

D2. Version control using git & GitHub

D3. Collaborative development using GitHub

D4. Test-driven development

D5. Continuous integration

Motivation

Thinking about costs

We have finite resources (time, energy, money) with which to conduct research. Tasks related to model development should use those resources efficiently.

Search & information costs

- How do I run the model? What does this line of code do?
- What about student/visitor X, who did Y two years ago—where is that version?
- What version of the model did I use to produce the results for paper X, which has been under review for 6 months?

Motivation

Policing & enforcement costs

- Who broke the model so Policy Z no longer has a feasible solution?
- When did our reference forecast start doing this weird thing in region *r* & sector *g*?
- What is our definitive 'reference' projection? Why does it differ between these two papers?
- Who changed this parameter & caused this reviewer to yell at me?

Recovery/disruption costs

The *bus factor* or *truck number*—"How many people would need to be hit by a truck, tomorrow, for us to suffer a serious setback in continuity/loss of work?"

C1. Reproducible research

C1. Reproducible research

Internal vs. external validity

Both are concerns in modeling & scenario research

Internal validity. Research is free of errors:

- Correctly implements methods w/o theoretical/conceptual errors.
- Confounding variables addressed to identify relationships between independent and dependent variables.
- Alternative hypotheses can be rejected.

External validity. Research is generalizable to other conditions:

- Research can be replicated or reproduced in a different study context.
- Research is robust to differences between the study context and other contexts to which conclusions are applicable.
- Research is robust to plausible alternatives to key assumptions.

What is a model? I

Three perspectives and resulting insights

A knowledge object that embodies or represents a theory or understanding of some real-world phenomenon.

- Theories often causal.
- Relationships expressed quantatively: equations connecting variables representing concepts measured in certain, systematic ways.
- Systematized concepts often aggregate: GDP, country, sector.

What is a model? II

Three perspectives and resulting insights

A scientific instrument that is used to perform experiments: “What would be the outcome (effect on quantity Y) if X were changed from x_1 to x_2?”

- Another important scientific instrument: the LHC.
    - EUR 7.5 billion budget; labour from many specialized roles.
    - Components for preparing the experiment, running it, measuring outcomes are carefully designed, constructed, tested.
- Instruments require meticulous attention to detail.
- Description of methods includes instruments so the experiment can be reproduced.

What is a model? III

Three perspectives and resulting insights

A software project in which people in organizations create code that is run on computer systems.

- All software has bugs; all organizations have politics.
- Software is constantly evolving and never complete.
- Tendency to overinvest time in code vis-à-vis documentation.
- “Technical debt”: code grows stale over time.
- Good software development practices are used to ensure that software meets needs.

Validity and reproducibility (r13y)

Since the model is not the real world, implications drawn from modeling results must be externally valid. Specific threats, as forms of uncertainty:

Structural. Is the theory a correct description of the phenomena?
$\rightarrow$ Responses: alternate model formulations.

Measurement uncertainty of input data and parameters.
$\rightarrow$ Sensitivity analyses, large ($> 10^3$) ensembles of model runs.

Epistemic uncertainty in conditions (e.g. future policy) that are unknowable, or whereof uncertainty cannot be quantified.
$\rightarrow$ Alternate scenarios.

All require a quality instrument that can be reused in an easy, automated manner, giving the same results every time—a **reproducible** model.

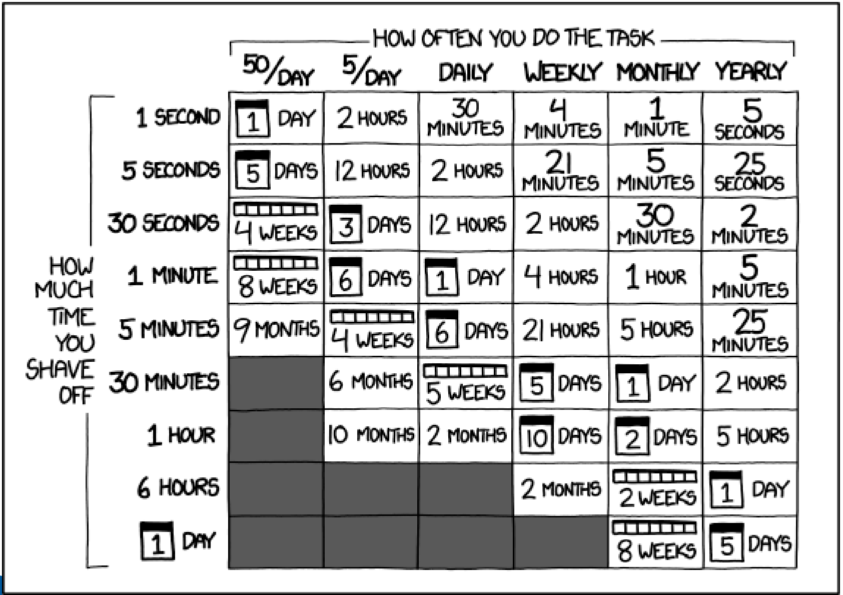
Modeling practice for validity & r13y

A collection of mutually-reinforcing practices:

- Version control.
- Collaborative development and documentation.
- Making software and data free and open in permanent archives.
- Internal & external peer review of modeling software.
- Manual or automated (continuous) testing.
- Organization and incentives to do the above.

Adopting best practices helps immensely, but is only part of ensuring validity and reproducibility. Other research tactics (e.g. careful *ex post* checks) also help—but they can be costly.

HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)



		HOW OFTEN YOU DO THE TASK					
		50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
HOW MUCH TIME YOU SHAVE OFF	1 SECOND	1 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
	5 SECONDS	5 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
	30 SECONDS	4 WEEKS	3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
	1 MINUTE	8 WEEKS	6 DAYS	1 DAY	4 HOURS	1 HOUR	5 MINUTES
	5 MINUTES	9 MONTHS	4 WEEKS	6 DAYS	21 HOURS	5 HOURS	25 MINUTES
	30 MINUTES	■	6 MONTHS	5 WEEKS	5 DAYS	1 DAY	2 HOURS
	1 HOUR	■	10 MONTHS	2 MONTHS	10 DAYS	2 DAYS	5 HOURS
	6 HOURS	■	■	■	2 MONTHS	2 WEEKS	1 DAY
	1 DAY	■	■	■	■	8 WEEKS	5 DAYS

Wrapping up: costs again

R13y in principle ≠ a habit of frequent reproduction.
If costs of reproduction are too high, people have little incentive to…

- often or ever try to reproduce results.
- identify, disclose, and correct internal *invalidity* (i.e. modeling errors).

When research methods are tacit knowledge rather than embodied in software, personnel changes make the cost of reproduction go up.

- Personnel changes are frequent in academia.

Conclusion → improve modeling practices in pursuit of reproducibility.

Further reading

- L. Barba group @ GWU SEAS: r13y syllabus w/readings on research group website; “The Hard Road to Reproducibility” in *Science*.
- Max Planck Institute for Meteorology “Good scientific practice” policy, rules, forms.
- Irving (2016), “A Minimum Standard for Publishing Computational Results in the Weather and Climate Sciences” in *Bulletin of the AMS*.
- Christensen & Miguel (2016), “Transparency, Reproducibility, and the Credibility of Economics Research” forthcoming in *JEL* — UC Berkeley Econ.
- Nick Barnes: “Publish your computer code: it is good enough” in *Nature News* — Climate Code Foundation.
- 45+ more peer-reviewed articles and other resources.

D. MESSAGEix development

Unit D: MESSAGEix development

This section presents an overview of the processes and tools used in development of the MESSAGE*ix* framework, ixmp, and the MESSAGEix-GLOBIOM model family of IIASA's Energy Program

References:

- `https://docs.messageix.org` $\rightarrow$ `v: master` (bottom-left) $\rightarrow$ "Contributing to MESSAGEix development"
- Existing content on GitHub, e.g. repositories `iiasa/message_ix`, `iiasa/ixmp`.

However, the concepts, tools, and processes are generalizable to any other model implemented as software.

D2. Version control using git & GitHub

D2. Version control using git & GitHub

Version control systems

Version control is the management of changes to documents, computer programs and other collections of information.

- Changes or states usually identified by a number or letter code.
- Each revision associated with a *timestamp* and *author*.
- Revisions can be *compared*, *restored*, and *combined*.

Version control systems (VCS, “revision control systems”, other names) are software that tracks and provide control over revisions.

- Automate repetitive, boring processes.
  - These could be (often are!) done manually.
  - But, because they are monotonous, mistakes are likely.
- Manage the chronological and sequential relationship between revisions.

`git`: a VCS

Several different VCS available. Some tools (e.g. Dropbox; MS Office "track changes") provides a subset of VCS-like features…but not suitable for models and scientific code.

We use `git` because it is popular, thus well-supported.

- A command-line (CLI) tool.
- Many GUI applications wrap around the CLI.
  E.g. GitHub Desktop, Atom editor, GitKraken.

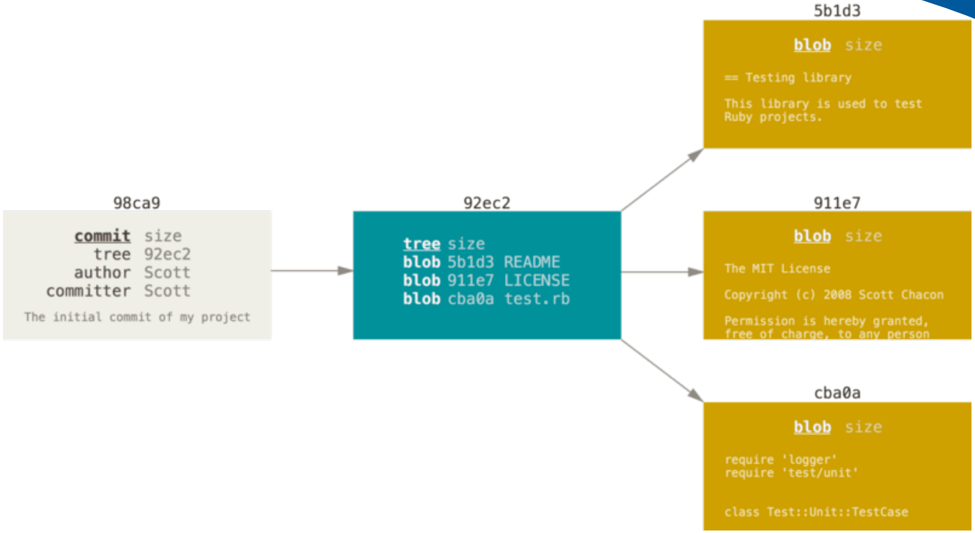
This lesson: a quick tour of key `git` concepts.

- *Many* more resources available online—search and find some; identify the ones most helpful to you.
- Here we use diagrams from the Git Book (available in 19+ languages).

`git` concepts: commit

A single version of a set of files arranged in directories.

- Author, timestamp, files (‘blobs’), description.
- ID or ‘hash’ e.g. `3f2ca4130cab262cfac62c5a98dd2ebdeb424dc5`.
    - We abbreviate with the first few characters: `3f2ca413`
- Hash of a previous (‘parent’) commit.
- ‘Snapshots’ of each file.



```
commit size
tree 92ec2
author Scott
committer Scott

The initial commit of my project
```

```
tree size
blob 5b1d3 README
blob 911e7 LICENSE
blob cba0a test.rb
```

```
blob size

== Testing library

This library is used to test
Ruby projects.
```

```
blob size

The MIT License

Copyright (c) 2008 Scott Chacon

Permission is hereby granted,
free of charge, to any person
```

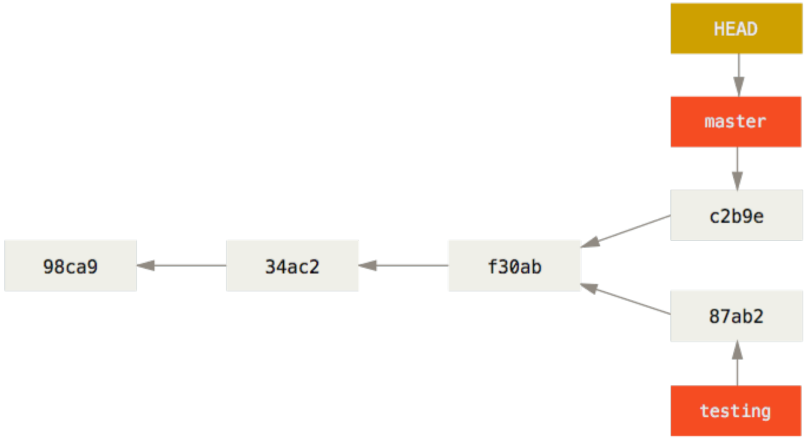
```
blob size

require 'logger'
require 'test/unit'


class Test::Unit::TestCase
```

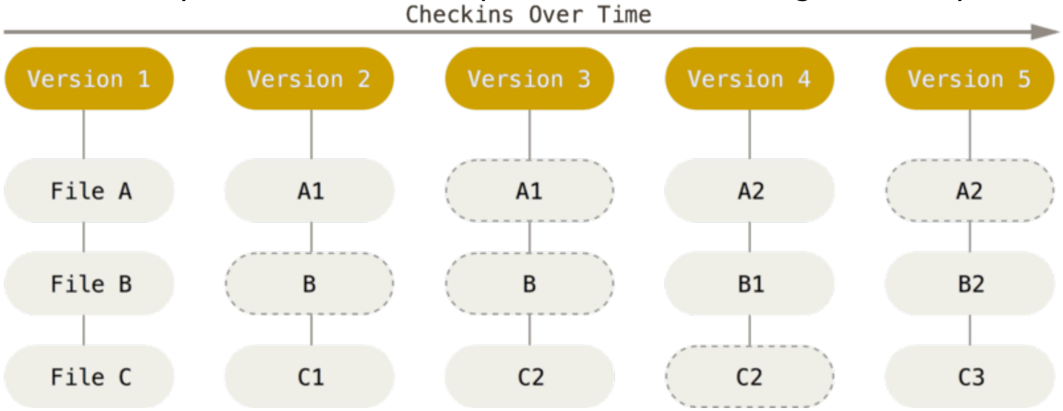
`git`: branch

A name for a particular commit *and* its ancestors:



git: branch

Commits may share the same snapshot of a file → storage efficiency.



`git` concepts: diff

Used to express changes between two snapshots of a single file:

Original file:

```
Shopping List

* Apples
* Oranges
* Salt
* Pepper
```

Modified file:

```
Shopping List
for Friday

* Apples
* Oranges (1 dozen)
* Salt
```

Changes

```
 Shopping List
+for Friday

 * Apples
-* Oranges
+* Oranges (1 dozen)
 * Salt
-* Pepper
```

`git` doesn't store these internally, but understands & generates them.

`git: tag`

A name applied to a certain commit.

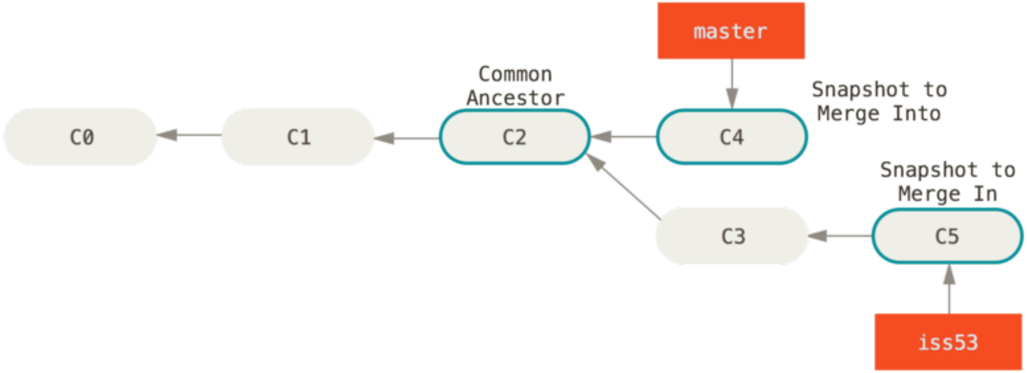
A *branch* can be extended by adding more commits to its head.
A *tag* always stays in the same place.

`git`: repository (‘repo’)

A collection of commits, snapshots, and tags.

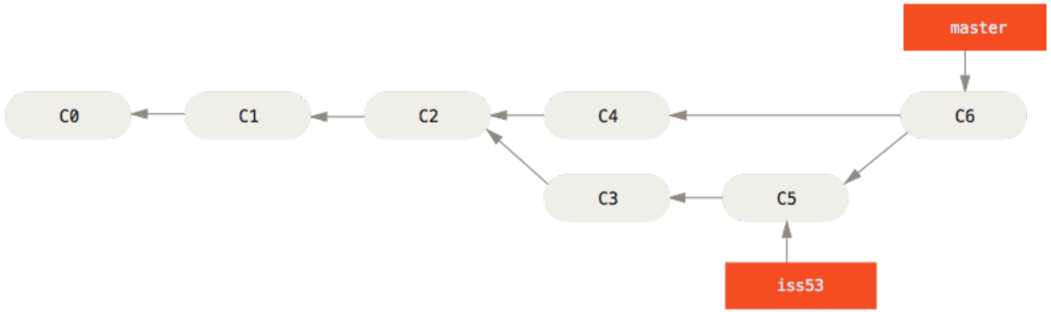
git: merge

Combines two commits from different branches.



git: merge

Creates a new commit.



`git`: merge

`git merge` automatically handles many tasks.

For example, changes to the same file:

- `branch-a` has a commit that modified `file.txt` near the top.
- `branch-b` has a commit that modified `file.txt` near the bottom.
- `git` applies *both* changes because they are non-overlapping, producing a combined `file.txt`

`git`: merge

Branch A changes:

```
Shopping List

 * Apples
-* Oranges
+* Oranges (1 dozen)
 * Salt
 * Pepper
```

Branch B changes:

```
Shopping List
+for Friday

 * Apples
 * Oranges
 * Salt
-* Pepper
```

Combined changes:

```
 Shopping List
+for Friday

 * Apples
-* Oranges
+* Oranges (1 dozen)
 * Salt
-* Pepper
```

→ keep files & directories neatly organized.

`git`: fetch/pull/push

`git` can *move commits* between *two* repos in different places:

- Two folders/directories on the same computer.
- Two computers: yours vs. a colleague's, or a server online.

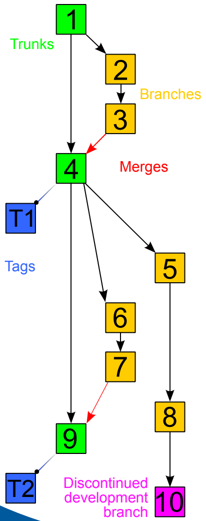
The other repo is called a remote. `git` helps you:

- Name and track multiple remotes related to the current repo.
- Associate a local branch with one branch on one remote.

Operations

- fetch: copy commits, branches, tags from a remote repo to yours. Doesn't change anything.
- pull: does three things
  1. Fetch a remote repo.
  2. *Add* new commits from the remote repo onto associated local branch.
  3. *Fast-forward* the pointer at the head of the local branch.
- push: pull, but in the opposite direction.

Another visualization



1, 2, ..., 10	commits.
2, 3	a branch; work done in parallel. Others can get & use 1 while 2, 3 are developed.
4, 9	merge commits. The changes made in 2, 3 (or 6, 7) are combined with 1 (or 4) to produce the new revision 4 (or 9).
1, 4, 9	the 'master' branch *Chosen* by the user to be the authoritative version of the code.
T1, T2	tags.

D3. Collaborative development using GitHub

D3. Collaborative development

General concepts

VCS like `git` provide *tools* for managing versions of code.

They do not:

- Require collaboration.
  You can use `git` in a single local repo without an Internet connection.
- Require that the files/code *do* anything, or be 'correct'.
- Prescribe *how* or *to what end* we should use them.

Software development comprises…

- the actions of conceiving, specifying, designing, programming, documenting, testing, and bug fixing…
- involved in creating and maintaining software.

General concepts

Collaborative development: when software development involves 2+ people embedded in 1+ organizations.

- Using a VCS can make this a lot easier, but…
- All involved must agree on *how* to use the VCS.

To collaborate, we must *communicate* about code:

- "[code] used to do X for me, but now it doesn't."
- "[code] says it will do X, but instead does Y."
- "[Al's code] does X, [Bo's code] does Y, but Jo wants to do both."
- "We fixed Y by making [changes] to [code]."
- "I wrote [new code] and I want everyone to use it."
- "You should use [version] instead of [version]."

GitHub

A (very) popular website.

You (user) or a group (organization) can store `git` repos on their servers.

More importantly, provides many tools for software development tasks (previous slide).

- These are tightly tied to specific `git` repos, branches, commits, and tags.
- They make it easy to use a certain workflow of software development.
- Understanding and using this workflow is a good basis for teams collaborating on software.

BUT (!)

GitHub's features are only *higher-level tools*, built on `git`.

They *suggest* a certain workflow, but every set of collaborators must still decide *whether* and *how* to use the features, and *what* their use means.

(!) below flags these decisions. For example:

- Alice and Bob both run into problems with Model X.
- Bob files a bug report (on GitHub) that doesn't prompt any action.
- Alice doesn't use GitHub at all. Her problem results in a new branch with many commits, lots of discussion, a quick merge into `master`, and a release—all via GitHub.

$\rightarrow$ Why did this happen?

GitHub workflow concepts: fork

A repo that is created by copying another repo.

Example:

- `https://github.com/iiasa` —IIASA organization.
- `https://github.com/iiasa/ixmp` —'main' repository for ixmp.
  - Can be made *public* or *private*.
  - View and push access can be controlled.
- `https://github.com/khaeru` —user profile.
- `https://github.com/khaeru/ixmp` —user's fork of ixmp.

Useful for working on changes for private use, or isolating work before it is merged with the main repo.
Can view all forks from a repo.

GitHub: release

A `git` tag with title, description, and associated downloads.

Example:

https://github.com/iiasa/ixmp/releases —all releases of ixmp.

GitHub: issue

A discussion about some bug, planned feature, or other issue (!) related to a specific repo.

Example: `https://github.com/iiasa/ixmp/issues/162`

- Identified by a number: `iiasa/ixmp#162`.
- Title and description from by the user who opened it; comments from others.
- Can be assigned to a particular user.
  (!) often the person responsible for fixing/addressing it.
- Can be associated with a label, milestone (later), or project (later).
- Status: open or closed. (!) Does 'closed' mean 'fixed'?
- `https://github.com/iiasa/ixmp/issues` —all issues for a repo.
  Search & filter tools.

GitHub: pull request (PR)

A request to `git` merge one branch into another (the 'base').

Example: `https://github.com/iiasa/ixmp/pull/309`

- Similar to issues: title, description, assignee(s), comments, label, milestone, project.
- Status: open, merged, or closed [without merging].
- Reviewer(s) — similar to assignees, 0+ other users (next slide).
- List of commits since the common ancestor.
- Collective diff for all changes introduced in the branch.
- Checks related to continuous integration tools (next lesson).

Caution: a branch named `iiasa:example` is not the same as `khaeru:example`!

GitHub: PR (continued)

Pull requests can *close* a specific issue, e.g. by fixing a bug or adding a desired feature.

Reviewers are requested, can view the commits and diff.

- Add comments on specific changed lines.
- Approve, request changes, or just comment.

(!) Collaborators must decide *how* to use PRs/reviews:

- Are reviews required? How many?
- Who can review the code?
- Different reviewers for different parts of code/types of issues or PRs?
- Should the code itself contain certain things?

`https://github.com/iiasa/ixmp/pulls` —all PRs for a repo.

GitHub: milestone

A target for collecting issues and pull requests.

Example:
`https://github.com/iiasa/message_ix/milestone/5?closed=1`

- Title and description.
- Status: open or closed.
- Can be assigned a target date.
- (!) What happens when the date passes?
- (!) Is a release created when the milestone is reached?

GitHub: project

Kanban-style system for organizing multiple tasks.

Example: `https://github.com/orgs/iiasa/projects/3`

- Cards for tasks that are either text (title/body) or links to issues/PRs.
- Columns that represent status of tasks.
- Automation to move cards when issues/PRs are created, closed, merged.
- Can bridge multiple repos.

D4. Test-driven development

D4. Test-driven development

C1. Reproducible research

D2. Version control using git & GitHub

D3. Collaborative development using GitHub

D4. Test-driven development
- Purpose
- Types of tests: unit, integration
- Test coverage
- `pytest`

D5. Continuous integration

Why test?

Software tests ensure that software meets quality standards.

Different kinds of tests ensure that…

- the code works as intended—in part and in whole;
- improvements do not cause *regressions*—breakage of existing features;
- speed, memory/storage use, and other performance metrics are within desired limits; and/or
- the software can be installed and used in its intended environment(s).

Testing can be done *manually* (following a list of instructions), but is most commonly *additional code* that:

1. Operates the target code in a certain way.
2. Checks outputs or performance against expected outcomes.

Types of tests

Unit tests of specific functions or lines—small pieces of code.

Integration tests of the interactions between lower-level components.

System tests of the completely integrated system.

Test-driven development:

1. Write tests *first*, as a way of saying:
   - “The code we write *should* do this.”
   - “The bug reported by Person A represents a failure to do this.”
2. Write or edit code until the tests pass.
3. Iterate as needed.

Test coverage

A metric that determines whether part or all of the software is tested.

Often measured as a number:
(number of lines of tested code) / (total lines of code)

Some common targets:

- Code changes/additions must not decrease overall test coverage.
- New additions must have 100% test coverage.
- Test coverage must be 100%.

Not the only important concept! Code may be 100% covered by unit-level tests, yet still fail to integrate or work correctly at higher, system level(s).

pytest

A Python software package that helps to write tests for other Python code. All parts of the MESSAGEix stack are tested using `pytest`.

Example:

```
def test_get_scalar(test_mp):
    scen = ixmp.Scenario(test_mp, *can_args)
    obs = scen.scalar('f')
    exp = {'unit': 'USD/km', 'value': 90}
    assert obs == exp
```

This tests (`assert obs == exp`) that the value returned by a certain function (`scen.scalar`) has a specific value (`exp`). If not, the text fails.

Success and failure for many tests is reported when `pytest` is run on the whole code base.

Learn `pytest`

Documentation: `https://docs.pytest.org` + many online examples

Discovery
: files and functions with names like `test_*.py` or `def test_foo(args):` are automatically collected and run.

Fixtures
: e.g. `test_mp` in the prior example: prepared Python objects used across multiple tests, generated by a function.

Configurable
: tests can be included or skipped based on command-line options, the operating system environment, etc.

Testing utilities: ixmp and other packages contain functions and tools that help test themselves *or* other software built on them.

Example: `ixmp.testing.make_dantzig()` sets up Dantzig's cannery/transport problem, as used in several other ixmp tests.

D5. Continuous integration

D5. Continuous integration

C1. Reproducible research

D2. Version control using git & GitHub

D3. Collaborative development using GitHub

D4. Test-driven development

D5. Continuous integration
- Purpose
- MESSAGE CI tools: GitHub Actions, RTD, Stickler, Codecov

Purpose of CI

Review from Lesson D4: “integration tests” confirm that lower-level components interact properly.

Continuous integration: frequent execution of integration and other tests.

- For atomic changes, i.e. individual commits, or according to a schedule, e.g. nightly.
- Performed by an automated system using test code and other configuration.

Why use CI?

To avoid rework.

- Running tests only at the *end* of a process of developing new code may turn up unexpected bugs or incompatibilities.
- This results in *rework*: re-writing code again to address these problems.

To be be robust to external conditions.

- Code that relies on e.g. an external web service or data source may be broken if that service/source changes.
- CI makes these issues visible immediately, so they may be fixed.

To enforce quality without human intervention.

This reduces the workload on pull request reviewers.

MESSAGE CI tools: GitHub Actions

Runs the test suite on Linux, macOS, Windows.

1. Watches a specific GitHub repo for new commits or PRs.
2. Starts 1+ *virtual machine(s)* with specific software.
   - e.g. multiple versions of Python.
3. `git fetch` the latest code.
4. Run a specific script defined in a file that lives with the code (`.github/workflows/`, in YAML format).
   - Scripts usually run the test suite, but also take other, configurable actions.
5. Build results left as a check on the associated PR.

Example for all tools: `https://github.com/iiasa/message_ix/pull/286`

CI tools: Read The Docs (RTD)

Builds (and hosts) the documentation.

- Documentation is stored as Markdown (`.md`) or ReStructuredText (`.rst`) files alongside the code (usually in `doc/source`).
- The Python program Sphinx (https://www.sphinx-doc.org/en/2.0/) turns these in HTML websites, PDF files and more.
- RTD…
  1. watches a repo (like GitHub Actions),
  2. uses Sphinx to build the docs, and then
  3. hosts them on the Web.
- Supports *multiple versions* for each repo—associated with branches.
- IIASA/ENE uses the commercial service to generate docs from *public/private* repos, use a custom domain docs.messageix.org, &c.

CI tools: Codecov

Analyses the coverage of tests run on other CI tools.

- `pytest-cov` plugin for `pytest` links it with the `coverage` package to measure coverage of lines/files.
- A *coverage report* is uploaded from GitHub Actions run to Codecov.
- Codecov provides a web interface for browsing reports,
- …compares PR code coverage with the coverage of the target (e.g. `master`) branch, and
- …leaves a check on GitHub if the PR maintains/improves coverage.

CI tools: Stickler

Checks for badly-formatted Python code.

- Leaves comments on specific lines as a GitHub reviewer.
- Tip: use a *linter* in your editor (e.g. `linter-flake8` for Atom) to ensure your code is clean before you commit and push.

CI tips and tricks

Earlier: one purpose of CI is "to enforce quality *without human intervention*."

However, CI is *even more useful* if used to help learn good development practice:

1. Before commit/push, a dev thinks, "When I push this/open a PR, will all the checks pass?"
2. "That includes GitHub Actions, AppVeyor (Windows tests)…"
3. "…but wait! I just added something that might only work on my Linux system."
4. "I'd better look again to make sure it works cross-platform, like other parts of the code."

$\rightarrow$ Steps 3 and 4 eventually become ingrained habit.

Wrapping up

Wrapping up

Always ask: Are development practices *clear*, and *clearly motivated*? If not, talk about it!

- Someone can explain it to you $\rightarrow$ ensure it's written down for others.
- It could prompt a conversation, reflection, and change in practice $\rightarrow$ better practice $\rightarrow$ better research.
- Written descriptions may not be up to date with current practice.

Read between the lines. What's *not* said in any org. is as important as what *is* emphasized.

- Editors, OS not mentioned today $\rightarrow$ we choose not to standardize.
- Scheduling releases.
- *Who* is assigned (or sees incentives) to work on *what*?
- Example of Alice and Bob's bugs—what tasks get priority? Why?

Thank you!