Train the trainer workshop

August/September 2024 CodeRefinery train the trainer workshop

Do you teach the use of computers and computational tools? Are you curious about scaling your training and learn about tested and proven practices from CodeRefinery workshops and other trainers? Join us for the CodeRefinery train the trainer workshop: four self-contained sessions on tools and techniques for computational training offer a great chance to enhance your teaching skills and learn about new tools and practices. What you will learn is also used a lot outside CodeRefinery, whenever good beginner friendly training is needed.

Learning objectives:

- Learn about tools and techniques and practice to create engaging online teaching experiences (screenshare, audio, etc.).
- Become mentally ready to be an instructor in a collaborative interactive online workshop (not only large workshops but in general).
- Learn how to design and develop lesson material collaboratively.

Target audience:

- Everyone teaching online workshops about computational topics or interested in teaching.
- Previous and future instructors and helpers of CodeRefinery workshops.

Prerequisites: An interest in teaching.

Workshop structure: We aim for a mix of modular presentations and work in small groups. The latter will be done in breakoutrooms - be prepared to switch on your camera and have a fruitful exchange with fellow participants!

Organizers and instructors:

The workshop is organized by partner organizations of the CodeRefinery project.

Facilitators and instructors:

- Radovan Bast
- Richard Darst

- Bjørn Lindi
- Dhanya Pushpadas
- Jarno Rantaharju
- Stephan Smuts
- Stepas Toliautas
- Samantha Wittke

Content and timing:

The workshop consists of four sessions every Tuesday between August 13th and September 3rd 2024, to :

- Session 1: About lesson design, deployment and iterative improvement (Aug 13)
- Session 2: Tools and techniques adopted in CodeRefinery workshops (Aug 20)
- Session 3: About teaching & cool things we all would like to share (Aug 27)
- Session 4: Workshop streaming practices and post-workshop tasks (Sep 3)

You can join all sessions, or the just the ones that interest you. More details on each session will be shared later.

The workshop is **free of charge for everyone**, please register below to get the Zoom link and other useful information for the workshop.

The workshop is over and registration is closed now.

If you have any questions, please write to support@coderefinery.org.

Lesson design and development

Objectives

- We share our design processes for teaching material and presentations.
- Learn how to design lessons “backwards”, starting from learning objectives and learner personas.
- Learn good practices for improving existing material based on feedback.

Instructor note

- Discussion: 20 min
- Exercises: 35 min

Exercise: How do you design your teaching material?

✍ We collect notes using a shared document (5 min)

- When you start preparing a new lesson or training material, where do you start?
- What tricks help you with “writer’s block” or the empty page problem?
- Maybe you haven’t designed training material yet. But how do you start when creating a new presentation?
- If your design process has changed over time, please describe what you used to do and what you do now instead.
- What do you know now about preparing lessons/training/presentations that you wish you knew earlier?

Creating new teaching material

Typical problems

- Someone creates a lesson, but they think about what is interesting to them, not what is important for the learners.
- “I want to show a number of things which I think are cool about tool X - how do I press these into 90 minutes?”
- Write down material you want to cover and then sprinkle in some exercises.
- Thinking about how I work, not how the learners work.
- Trying to bring learners to their level/setup, not trying to meet the learners where they are.
- Not really knowing the learning objectives or the learner personas.

Better approach

Good questions to ask and discuss with a group of colleagues **from diverse backgrounds**:

- What is the expected educational level of my audience?
- Have they been already exposed to the technologies I am planning to teach?
- What tools do they already use?
- What are the main issues they are currently experiencing?
- What do they need to remember/understand/apply/analyze/evaluate/create (Bloom’s taxonomy)?
- Define learner personas.
- It may be an advantage to share an imperfect lesson with others early to collect feedback and suggestions before the lesson “solidifies” too much. Draft it and collect feedback. The result will probably be better than working in isolation towards a “perfect” lesson.

The process of designing a lesson “backwards”

As described in "A lesson design process" in the book Teaching Tech Together:

1. Understand your learners.
2. Brainstorm rough ideas.
3. Create an summative assessment to know your overall goal.

    [Think of the things your learners will be able to do at the end of the lesson]

4. Create formative assessments to go from the starting point to this.

    [Think of some engaging and active exercises]

5. Order the formative assessments (exercises) into a reasonable order.
6. Write just enough material to get from one assessment (exercise) to another.
7. Describe the course so the learners know if it is relevant to them.

Improving existing lessons

💬 All CodeRefinery lessons are on GitHub

- Overview: https://coderefinery.org/lessons/
- All are shared under CC-BY license and we encourage reuse and modification.
- Sources are all on GitHub: https://github.com/coderefinery
- Web pages are generated from Markdown using Sphinx (more about that in the episode Lessons with version control).
- We track ideas and problems in GitHub issues.

Collect feedback during the workshop:

- Collect feedback from learners and instructors (Example from a past workshop).
- Convert feedback about lessons and suggestions for improvements into issues so that these don't get lost and stay close to the lesson material.

Collect feedback before you start a big rewrite:

- First open an issue and describe your idea and collect feedback before you start with an extensive rewrite.
- For things still under construction, open a draft pull/merge request to collect feedback and to signal to others what you are working on.

Small picture changes vs. big picture changes:

- Lesson changes should be accompanied with instructor guide changes (it's like a documentation for the lesson material).
- Instructor guide is essential for new instructors.
- Before making larger changes, talk with somebody and discuss these changes.

Use case: our lessons

As an example to demonstrate the process of designing and improving lessons, we will have a look at one of our own lessons: Introduction to version control with Git.

- Initial 2014-2016 version
  - https://github.com/scisoft/toolbox-talks and https://toolbox.readthedocs.io/
  - Amazingly they are still findable!
  - Format: Slides and live coding.
  - Exercises were separate, during afternoon sessions.
- Some time in 2014-2015 attended Carpentries instructor training.
- 2016: CodeRefinery started.
- 2017: Started a new repository based on the Carpentries lesson template (at the time using Jekyll).
  - Exercises become part of the lesson.
  - We start in the **command line** and only later move to GitHub.
- 2019: A lot more thought about learning objectives and personas.
  - Also license change to CC-BY.
- 2022: Convert lesson from Jekyll to Sphinx.
  - Using the tools that we teach/advocate.
  - We can have tabs and better code highlighting/emphasis.
  - Easier local preview (Python environment instead of Ruby environment which we were not used to in our daily work).
- 2024: Big redesign. We move the lesson closer to where learners are.
  - Start from GitHub instead of on the command line.
  - Start from an existing repository instead of with an empty one.
  - Offer several tracks to participate in the lesson (GitHub, VS Code, and command line) and learners can choose which one they want to follow.
  - Blog post: We have completely changed our Git lessons. Hopefully to the better.
- Next steps?
  - Making the lesson citable following our blog post.
  - Improvements based on what we learn from this workshop.

The overarching trend was to make the lesson simpler and more accessible - to meet the learners where they are instead of pulling them to the tool choices of the instructors. Looking back, we learned a lot and the learning process is not over yet.

Exercise: Discussion about learning objectives and exercise design

✍ We work in groups but use the shared document as result (20 min)

1. As a group **pick a lesson topic**. It can be one of the topics listed here but you can also choose something else that your group is interested in, or a topic that you have taught before or would like to teach. Some suggestions:
   - Git: Creating a repository and porting your project to Git and GitHub

- Git: Basic commands
- Git: Branching and merging
- Git: Recovering from typical mistakes
- Code documentation
- Jupyter Notebooks
- Collaboration using Git and GitHub/GitLab
- Using GitHub without the command line
- Project organization
- Automated testing
- Data transfer
- Data management and versioning
- Code quality and good practices
- Modular code development
- How to release and publish your code
- How to document and track code dependencies
- Recording environments in containers
- Profiling memory and CPU usage
- Strategies for parallelization
- Conda environments
- Data processing using workflow managers
- Regular expressions
- Making papers in LaTeX
- Making figures in your favorite programming language
- Linux shell basics
- Something non-technical, such as painting a room
- Introduction to high-performance computing
- A lesson you always wanted to teach
- …

2. Try to define 2-3 learning objectives for the lesson and write them down. You can think of these as "three simple enough messages that someone will remember the next day" - **they need to be pretty simple**.
3. Can you come up with one or two engaging exercises that could be used to demonstrate one of those objectives? **They should be simple** enough people can actually do them. Creating simple exercises is not easy. Some standard exercise types:
   - Multiple choice (easy to get feedback via a classroom tool - try to design each wrong answer so that it identifies a specific misconception).
   - Code yourself (traditional programming)
   - Code yourself + multiple choice to see what the answer is (allows you to get feedback)
   - Inverted coding (given code, have to debug)
   - Parsons problems (working solution but lines in random order, learner must only put in proper order)
   - Fill in the blank
   - Discussions, self directed learning exercises

Great resources

- Teaching Tech Together
- Our summary of Teaching Tech Together
- Ten quick tips for creating an effective lesson
- Carpentries Curriculum Development Handbook
- Our manual on lesson design

Lessons with version control

Objectives

- Understand why version control is useful even for teaching material
- Understand how version control managed lessons can be modified.
- Understand how the CodeRefinery lesson template is used to create new lessons

Instructor note

- Discussion: 25 min
- Exercises or demos: 20 min

Why version control?

- If you are in CodeRefinery TTT, you probably know what version control is and why it is important.
- The benefits of version control also extend to lessons:
  - Change history
  - Others can submit contributions
  - Others can make derived versions and sync up later
  - Same workflow as everything else
  - Write it like documentation: probably more reading after than watching it as a presentation.
- Disadvantages
  - “What you see is what you get” editing is hard
  - Requires knowing version control

💬 Accepting the smallest contribution

Question: if someone wants to make a tiny fix to your material, can they?

Tour of lesson templates options

There are different ways to make lessons with git. Some dedicated to teaching:

- CodeRefinery
  - Example: This lesson itself
  - Based on the Sphinx documentation generator
  - sphinx-lesson is *very* minimal extra functionality
- Carpentries
  - Example: https://carpentries.github.io/lesson-example/
  - Based on R and Rmarkdown

Our philosophy is that anything works: it doesn't have to be just designed for lessons

- Jupyter Book
  - Example: https://jupyterbook.org/
  - Note: is based on sphinx, many extensions here are used in CR lessons
- Various ways to make slides out of Markdown
- Cicero: GitHub-hosted Markdown to slides easily
  - Demo: Asking for Help with Supercomputers The source
- Whatever your existing documentation is.

We like the CodeRefinery format, but think that you should use whatever fits your needs the most.

Sphinx

- We build all our lesson material with Sphinx
- Generate HTML/PDF/LaTeX from RST and Markdown.
- Many Python projects use Sphinx for documentation but **Sphinx is not limited to Python**.
- Read the docs hosts public Sphinx documentation for free!
- Also hostable anywhere else, like Github pages, like our lesson material
- For code a selling point for Sphinx is that also API documentation is possible.

Sphinx is a doc generator, not HTML generator. It can:

- Markdown, Jupyter, and ReST (and more...) inputs. Executable inputs.
  - jupyter-book is Sphinx, so anything it can do we can do. This was one of the inspirations for using Sphinx
- Good support for inline code. Much more than static code display, if you want to look at extensions.
- Generate different output formats (html, single-page html, pdf, epub, etc.)
- Strong cross-referencing within and between projects

CodeRefinery lesson template

It is "just a normal Sphinx project" - with extensions:

- Sphinx lesson extension
  - adds is various directives (boxes) tuned to lesson purposes

    - provides a sample organization and template repo you can use so that lessons look consistent
- Sphinx gives us other nice features for free
    - Tabs: they are very important for us and allow us to customize in-place instead of copying everything and fragmenting lessons
    - Emphasize lines: they make it easier to spot what has changed in longer code snippets
    - Various input formats
        - Markdown (via the MyST-parser), ReStructured text, Jupyter Notebooks.
    - Many other features designed for presenting and interacting with code
- It's fine if you use some other static site generator or git-based lesson method.

👀 Instructors go through the building and contributing process

Depending on the course, instructors will demo what is roughly exercise 4 below. Or a course might go straight to exercises.

- Instructors decide what change they would want to make
- Instructors clone the repository
- **Instructors make the change**
- **Instructors set up the build environment**
- **Instructors build and preview**
- Instructors command and send upstream

Exercises

Some exercises have prerequisites (Git or Github accounts). Most instances of this course will have you do **1 and 2** below.

✍️ Lesson-VCS-1: Present and discuss your own lesson formats

We don't want to push everyone towards one format, but as long as you use Git, it's easy to share and reuse.

- Discuss what formats you do use
- Within your team, show examples of the lessons formats you use now. Discuss what is good and to-be-improved about them.
- Look at how they source is managed and how easy it might be to edit.

✍️ Lesson-VCS-2: Tour a CodeRefinery or Carpentries lesson on Github

- Look at either a CodeRefinery or Carpentries lesson
    - CodeRefinery Git-Intro: Lesson, Github repo
    - Carpentries Linux shell: Lesson, Github repo
- Can you find
    - Where is the content of the lessons?
    - What recent change propsals (pull requests) have been made?

- What are the open issues?
- How you would contribute something?
- How would you use this yourself?

✍ Lesson-VCS-3: Modify a CodeRefinery example lesson on Github

In this, you will make a change to a lesson on Github, using only the Github interface. (Github account required, and we assume some knowledge of Github. Ask for help in your team if it is new to you!)

- Navigate to the example lesson we have set up: repo, web
- Go to some page and follow the link to go to the respective page on Github. (Alternatively, you can find the page from the Github repo directly).
- Follow the Github flow to make a change, and open a Pull Request with the change proposal:
  - Click on the pencil icon
  - Make a change
  - Follow the flow to make a commit and change. You'll fork the repository to make your own copy, add a message, and open a pull request.

We will look at these together and accept some changes.

✍ Lesson-VCS-4: Clone and build a CodeRefinery lesson locally

In this exercise, you will use Git to clone one a CodeRefinery lesson and try to preview it locally. It assumes installation and knowledge of Git.

- Use this sample repository: git-intro (or whatever else you would like)
- Clone the repository to your own computer
- Create a virtual environment using the `requirements.txt` contained within the repository.
- Build the lesson.
  - Most people will probably run: `sphinx-build content/ _build/`
  - If you have `make` installed you can `make html`
  - Look in the `_build` directory for the built HTML files.

Overall, this is pretty easy and straightforward, if you have a Python environment set up. The CodeRefinery documentation lesson teaches this for every operating system.

This same tool can be used to build documentation for other software projects and is pretty standard.

✍ Lesson-VCS-5: (advanced) Create your own lesson using the CodeRefinery format

In this lesson, you'll copy the CodeRefinery template and do basic modifications for your own lesson.

- Clone the lesson template: https://github.com/coderefinery/sphinx-lesson-template
- Attempt to build it as it is (see the previous exercise)
- How can you do tabs?
- How can you highlight lines in code boxes?
- How can you change color, logo and fonts?
- What directives are available?

Summary

Keypoints

- Version control takes teaching materials to the next level: shareable and easy to contribute
- There are different formats that use version control, but we like Sphinx with a sphinx-lesson extension.

How we collect feedback and measure impact

Objectives

- Discuss how one can collect feedback from learners ("what can we improve?").
- Discuss how we convert feedback into actionable items.
- Discuss how we measure the impact of teaching ("did we achieve our goals?").
- Discuss the "why".
- Get to know the reasons and sources of inspiration behind major lesson and workshop updates.

Instructor note

- Discussion: 20 min
- Exercises: 10 min

Asking questions before the workshop

- Motivation: Know your audience.
- Until 2021 we had a pre-workshop survey:
  - Data, questions, and notebook: https://github.com/coderefinery/pre-workshop-survey/
  - Zenodo/DOI: https://doi.org/10.5281/zenodo.2671578
- After 2021 we incorporated some of the questions into the registration form.
  - Easier registration experience for participants.
  - After 5 years of running workshops, we had a good idea of what to expect.

Collecting feedback as we teach

- Each day we ask for feedback in the collaborative notes.
  - One good thing about today.
  - One thing to improve for next time.
  - Any other comments?
- During the workshop we sometimes check in and ask about the pace, example:

```
How is the speed so far? (add an "o")
- Too fast:   oooooo
- Too slow:   ooo
- Just right: oooooooooooooooooo
```

- We publish all questions, answers, and feedback. Example: https://coderefinery.github.io/2024-03-12-workshop/questions/
- How we follow up:
  - Some problems we can fix already before the next workshop day.
  - We convert feedback/problems into GitHub issues and track these close to the lesson material.

Trying to measure impact with longer-term surveys

- Motivation: Understand the long-term impact of our workshops. Have something to show to funders.
- 2024 post-workshop survey:
  - Blog post: https://coderefinery.org/blog/2024/08/10/post-workshop-survey/
  - Questions, notebook, and figures: https://github.com/coderefinery/2024-post-workshop-survey
  - Zenodo/DOI: https://doi.org/10.5281/zenodo.13292363
- 2021 version:
  - Data, questions, notebook, and figures: https://github.com/coderefinery/post-workshop-survey
  - Zenodo/DOI: https://doi.org/10.5281/zenodo.2671576
- How we use the results:
  - When reporting to funders.
  - When planning future workshops and bigger picture changes.

Lessons learned

- Think about how to measure impact/success from the beginning.
- **Make the feedback and survey results public**.
- Make the results persistent and citable.
- Have a mechanism to follow-up on feedback.

- Anonymization is more than just removing or dissociating names.
- Take time designing your survey and collect feedback on the survey itself.

Take time designing your survey

We are not experts in survey design but we reached out to RISE Research Institutes of Sweden to get feedback on our survey questions. They provided us with a lot of valuable feedback and suggestions for improvements. Below are few examples.

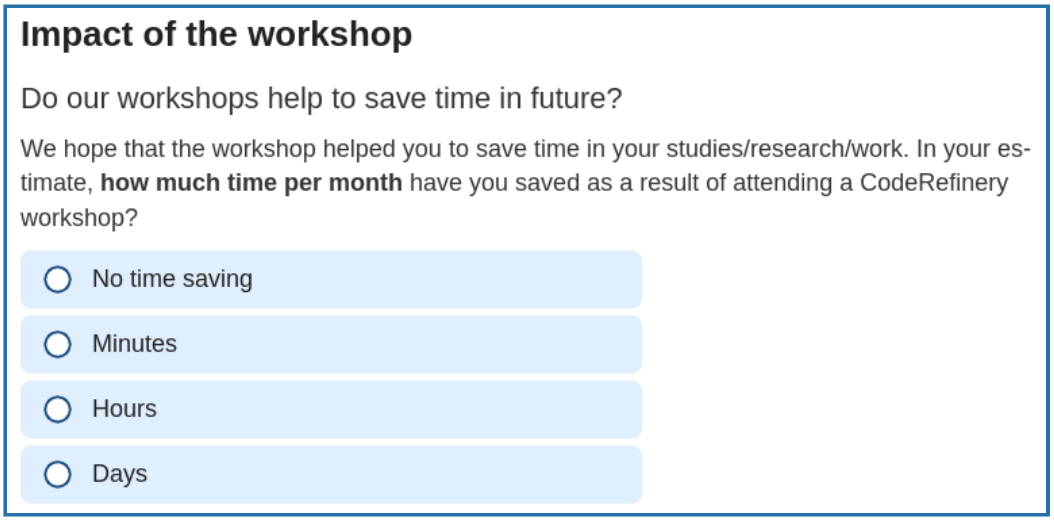
Example 1

First version of our survey question about impact:

- “Do our workshops help to save time in future?” Please quantify in hours saved: …

Feedback:

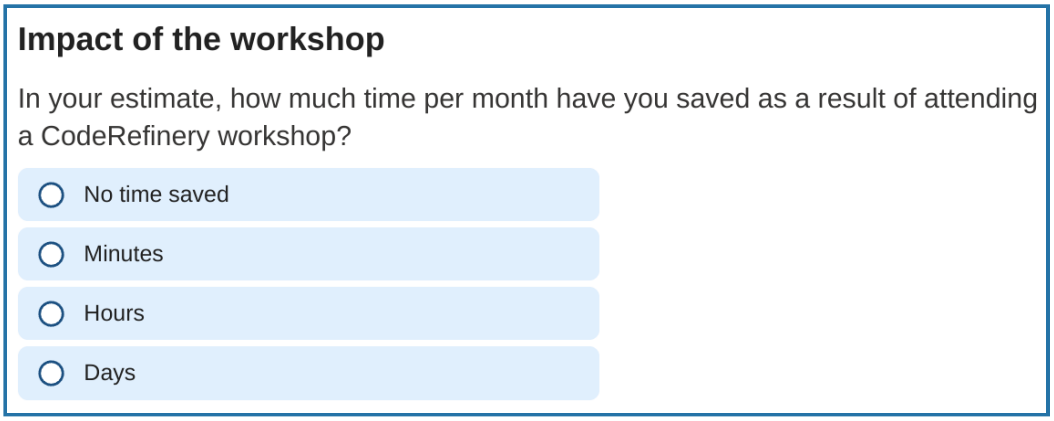
- Difficult to answer.
- Since when? Until all eternity?



Earlier version of the survey question about impact.

Feedback:

- The question “Do our workshops help to save time in future?” is unspecified, unnecessary, and leading.
- “No time saving” does not match the wording “save time” in the question.



Later version of the survey question about impact.

- The wording “have you saved” now matches “No time saved”.

Example 2

- Earlier version:

```
Would you judge your code to be better reusable/reproducible/modular/documented
as a result of attending the workshop?

- More reusable
- More reproducible
- More modular
- Better documented
- None of the above
```

- Feedback: The question is not neutrally formulated and risks leading to over-reporting of yes answers. Consider balancing so that the questions are formulated more neutrally.
- Reformulated to:

```
After attending the workshop, would you judge your code to be
more reusable or not more reusable?

- My code is more reusable
- My code is not more reusable
- Not sure
```

Example 3

- Early version:

```
What else has changed in how you write code since attending the workshop?

[free-form text field]
```

- Feedback: Leading and assumes that something has changed.
- Reformulated to:

```
Has anything else changed in how you write code for your research after
attending the workshop?

[free-form text field]
```

Example 4

- Earlier version:

```
Has it become easier for you to collaborate on software development with your
colleagues and collaborators?

- Yes
- Not sure
- No
```

- Feedback:
  - Leading question.
  - Avoid “Yes” and “No” response options because respondents tend to answer “Yes” if that option is available, leading to a risk of measurement error (acquiescence bias).
  - Do not place “Not sure” in the middle of the scale because it captures participants who actually don’t know and should therefore be placed at the bottom instead of in the middle of the scale.
- Reformulated to:

```
After attending the workshop, has it become easier or not for you to
collaborate on software development with your colleagues and collaborators?

- Collaboration is easier
- Collaboration is not easier
- Not sure
```

Exercise: Group discussion (10 min)

✍ Group discussion using the collaborative notes

- What tricks/techniques have you tried in your teaching or seen in someone else’s teaching that you think have been particularly effective in collecting feedback from learners?
- Can you give tips or share your experiences about how to convert feedback into actionable items?
- How do you measure the impact of your teaching? Any tips or experiences about what you have tried or seen other courses do?
- Anybody knows of good resources on survey design? Please link them in the collaborative notes.

About the CodeRefinery project and CodeRefinery workshops in general

Objectives

- Discuss what CodeRefinery is and how we got here
- Understand about the challenges to define our target audience

CodeRefinery is a Nordic e-Infrastructure Collaboration (NeIC) project that has started in October 2016 and is currently funded until February 2025. We are working on the continuation plans.

The funding from 2022-2025 is designed to keep this project active beyond 2025 by forming a support network and building a community of instructors and contributors.

History

The CodeRefinery project idea grew out of two SeSE courses given at KTH Stockholm in 2014 and 2016. The project proposal itself was submitted to NeIC in 2015, accepted in 2015, and started in 2016.

We have started by porting own lessons to the Carpentries teaching style and format, and collaboratively and iteratively grew and improved the material to its present form.

Goals

- Develop and maintain **training material on good enough software development practices** for researchers that write code/scripts/notebooks.
- Our material addresses all academic disciplines and tries to be as programming language-independent as possible.
- Provide a code repository hosting service that is open and free for all researchers based in universities and research institutes from Nordic countries.
- Provide **training opportunities** in the Nordics using (Carpentries and) CodeRefinery training materials.
- Evolve the project towards a **community-driven project** with a network of instructors and contributors.

Community

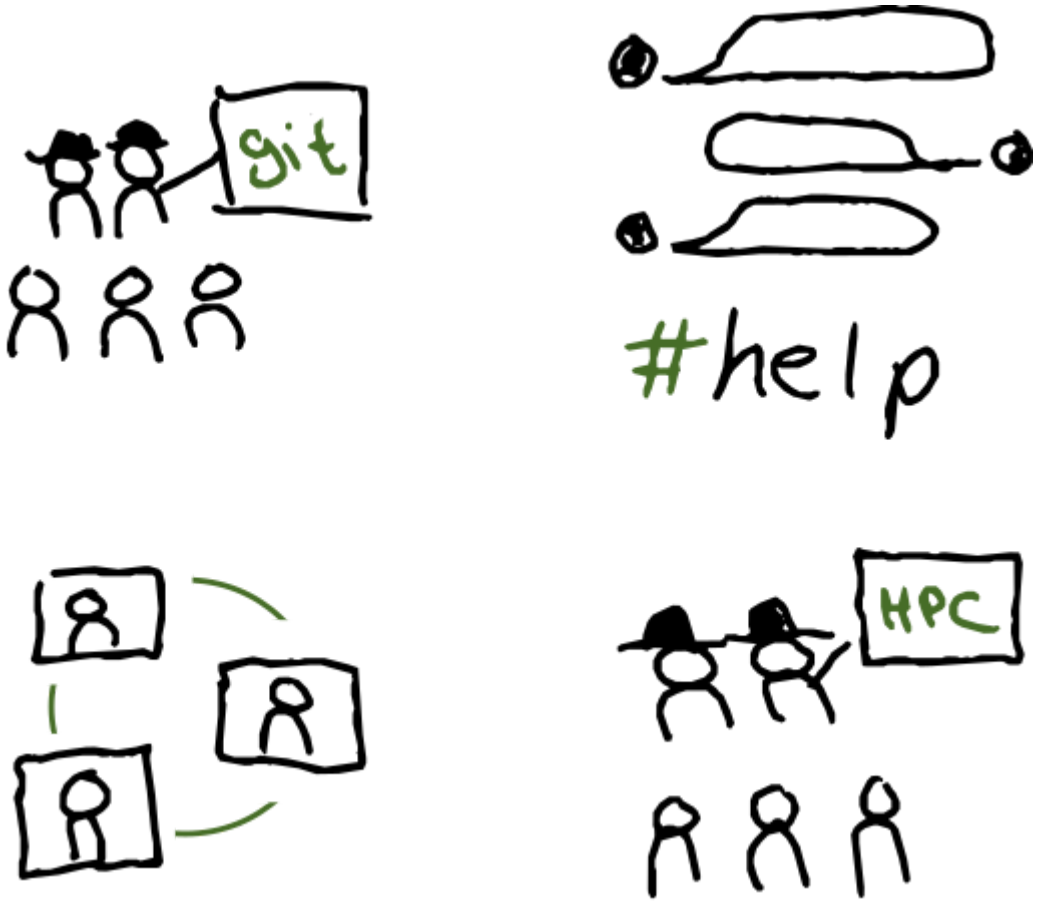


Image showing the key areas of the CodeRefinery community: Workshops, chat inlcuding help channel, online meetings and co-working, other collaborative training, eg on High Performance Computing topics.

CodeRefinery is not just workshops, we are community and want you to be part of it!

There are many different levels of involvement, from the occasional commenter in chat, CodeRefinery ambassador (people who like the project and workshops and help us spreading the word) or lesson issue creators and lesson contributors to local host, co-instructor or co-organizer.

Best **first step** in any case is to join the CodeRefinery Zulip chat or let us know about your interest at support@coderefinery.org.

Target audience

One common question we get is how do we relate to the Carpentries. This section describes how we see it:

Carpentries audience

The Carpentries aims to teach computational **competence** to learners through an applied approach, avoiding the theoretical and general in favor of the practical and specific.

Mostly, learners do not need to have any prior experience in programming. One major goal of a Carpentries workshop is to raise awareness on the tools researchers can learn/use to speed up their research.

By showing learners how to solve specific problems with specific tools and providing hands-on practice, learners develops confidence for future learning.

Novices

We often qualify Carpentries learners as **novices**: they do not know what they need to learn yet. A typical example is the usage of version control: the Carpentries lesson Version Control with Git aims to give a very high level conceptual overview of Git but it does not explain how it can be used in research projects.

CodeRefinery audience

In that sense, CodeRefinery workshops differ from Carpentries workshops as we assume our audience already writes code and scripts and we aim at teaching them **best software practices**.

Our learners usually do not have a good overview of **best software practices** but are aware of the need to learn them. Very often, they know of the tools (Git, Jupyter, etc.) we are teaching but have difficulties to make the best use of them in their software development workflow.

Whenever we can, we direct learners that do not have sufficient coding experience to Carpentries workshops.

Competent practitioners

We often qualify CodeRefinery learners as **competent practitioners** because they already have an understanding of their needs.

💬 Challenges related to defining our target audience

We often get the feedback "I wish I would have known X earlier!" *Competent practitioners* have run into issues with **not** caring (or not fully understanding) about version control, documentation, modularity, reproducibility before, so they are easily motivated to learn more.

For a *novice* these topics may seem unnecessary and "too much" and the workshop may feel too difficult to follow. However, the materials are designed so that one can always revisit a topic, when needed. The important part is that you know that "X" exists, and where to find more information, which is also beneficial for novices.

Keypoints: CodeRefinery

- Teaches intermediate-level software development tool lessons
- It is difficult to define “best practices”, we try to teach **“good enough” practices**
- Training network for other lessons
- Publicly-funded discrete projects (3 projects actually) transitioning towards an open community project
- We have online material, teaching, and exercise sessions
- Our main target audience are competent practitioners, but also novices and experts can get something out of the workshops
- We want more people to work with us, and to work with more people

Collaborative notes

Objectives

- Be able to provide a highly interactive online workshop environment with collaborative documents

Instructor note

- Teaching: 10 min
- Exercise: 15 min
- Questions & Answers: 5 min

Introduction

The Collaborative document is how you interact with the participants. The participants can ask questions and give feedback through the collaborative document. During a CodeRefinery session there can be a large a volume of questions. A dedicated person, a Collaborative document-manager, is needed to answer and edit the collaborative document. Let us see how the collaborative document is used, then discuss the role of the editor or collaborative document-manager.

Collaborative document mechanics and controls

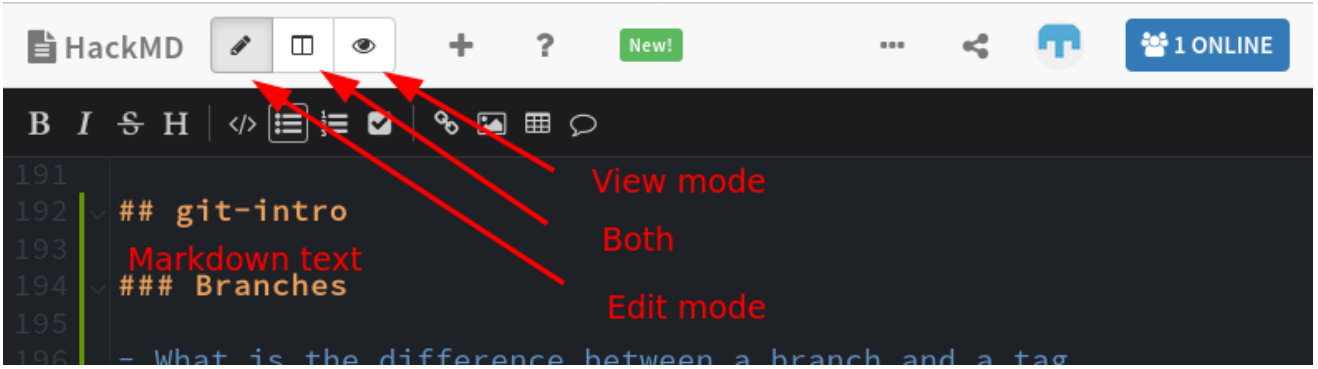
Technologies that can be used as a collaborative document are Hackmd, HedgeDoc, or Google Docs

Hackmd or HedgeDoc are real-time text editor online. We use it to:

- As a threaded chat, to **answer questions and provide other information** without interrupting the main flow of the room.
- provide everyone with a **more equal opportunity to ask questions**.
- **create notes** which will be archived, for your later reference.

You do not need to login/create an account to be able to edit the document.

Basic controls

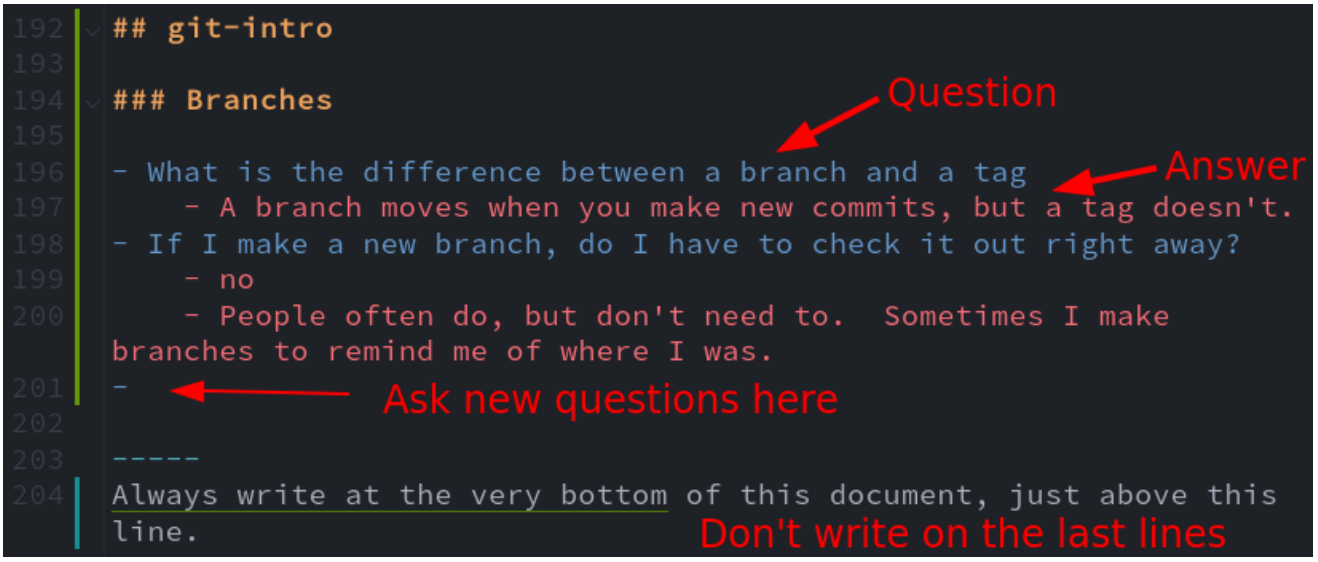


This may look slightly different on mobile devices and small windows.

- At the top (left or right), you can switch between **view**, **edit**, and **split view and edit** modes.
- You write in markdown here. Don't worry about the syntax, just see what others do and try to be like that! Someone will come and fix any problems there may be.
- Please go back to view mode if you think you won't edit for a while - it will still live update.

Asking questions

Always ask questions and add new sections at the very bottom. You can also answer and comment on older questions, too.



Questions and answers in bullet points

Since we plan to publish the questions and answers later as part of the workshop page, we recommend to not use any names. You can indicate your own name to make it easier to discuss more during the workshop but then always use this form: `[name=Myname]`. This makes it easier for us to automatically remove all names before publishing the notes.

Other hints:

- Use `+1` to agree with a statement or question (we are more likely to comment on it).
- Please leave some blank lines at the bottom
- NOTE: Please don't "select all", it highlights for everyone and adds a risk of losing data (there are periodic backups, but not instant).
- It can be quite demanding to follow the collaborative document closely. Keep an eye on it, but consider how distracted you may get from the course. For things beyond the scope of the course, we may come back and answer later.

Don't get overwhelmed

There can be a flood of information on the collaborative document. Scan for what is important, then if you would like come back later. But it is important to keep checking it.

Privacy

- Assume the collaborative document is **public and published: you never need to put your name there**.
- The collaborative document will be **published on the website afterwards**. We will remove all non-instructors names, but it's easier if you don't add it there in the first place.
- Please keep the link private during the workshop, since since security is "editable by those who have the link".
- You can use `[name=YOURNAME]`, to name yourself. We *will* remove all names (but not the comments) before archiving the notes (use this format to make it easy for us).

Exercise

✍ Discuss how to collaborate and handle questions (15 min)

Write down your conclusions in the shared document. Items to discuss are:

- What is your experience with questions and discussions while teaching?
- How do you deal with them?
- What kind of technologies do you prefer: chat, shared document, or voices and discussion raised during instruction? And why?

Collaborative Document Manager

We have one person who is a "Collaborative Document helper". This isn't the only person that should edit and answer, but one person shouldn't have too much else on their mind so can focus on it. They also make sure that the collaborative document is updated with exercise, break, and other meta-information to keep people on track.

Below, (*) = important.

Before the workshop

- Create a new collaborative document for the workshop
- make sure that **editing is enabled for anyone without login**
- Add workshop information, links to the workshop page and material and an example question and answer to the top of the collaborative document(see below)

Most things to edit (everyone)

Make it easy to post after the course and consistent to follow:

- Tag all names with `[name=XXX]` (so they can be removed later), remove other personal data or make it obvious.
- Add in information on exercises (new section for them, link, end time, what to accomplish)
- Make a logical section structure (`#` for title, `##` for sections, `###` for episodes, etc. - or what makes sense)

General Collaborative Document practices

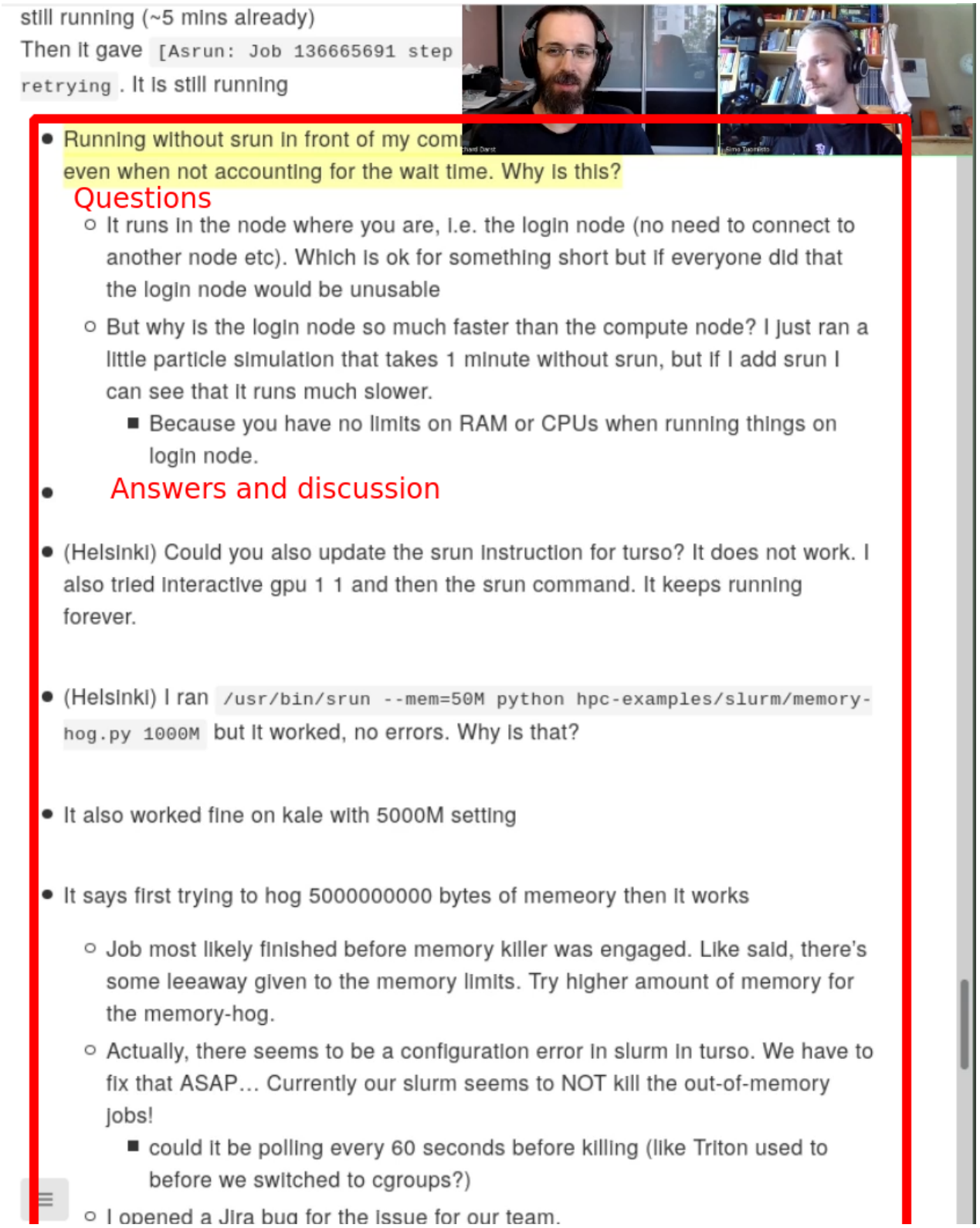
Keep it formatted well:

- (*) Tag names you see with `[name=XXX]` so that we can remove it later.
- Heading level `#` is only the page title
- Add a new `##` heading when a new *lesson* or similar thing is started (introduction, icebreaker, break between lessons, etc)
- Add a new `###` heading when a new *episode*, *exercise*, *break* (within exercise session)
- Ensure people are asking questions at the bottom, direct them there if they aren't.
- (*) Ensure each question is a bullet point. Each answer or follow-up should be a bullet point below.
    - Should you use more deeply nested bullet points, or have only one level below the initial question? It depends on the context, but if a conversation goes on too long, try not to let it go too deep.

Update with meta-talk, so that learners can follow along easily:

- Add Icebreaker and introductory material of the day. Try to talk to people as they joined to get them to open the collaborative document and answer.
- Anything important for following along should not be only said via voice. It needs to be in the collaborative document, too.
- New lessons or episodes, with links to them.
- For exercises, link to exercise and add the duration, end time, goals. If these are unclear, bring it up to the instructor by voice.
- Add a status display about breaks.

Screenshare it when necessary:



A live demo of a Collaborative Document during a Q&A time. The two instructors are discussing some of the import answers. Multiple learners have asked questions, multiple answers, and some remaining to be answered

- During breaks and other times, share the collaborative document(including the notification about break, and when it ends).
- It is nice if the arrangement allows some of the latest questions to be seen, so people are reminded to ask there.
- Someone else may do this, but should make sure it happens.

Answer questions

- If there is an question that should be answered by the instructor by voice, bring it up (by voice) to the instructor immediately.
- How soon do you answer questions? Two points of view:
  - Answer questions right away: can be really intense to follow.
  - Wait some so that we don't overload learners: reduces the info flow. But then do people need to check back more often.
  - You need to find your own balance. Maybe a quick answer right away, and more detailed later. Or delay answers during the most important parts of the lecture.
- Avoid wall-of-text answers. If reading an answer takes too long, it puts the person (and other people who even try to read it) behind even more by taking up valuable mental energy. If an answer needs a wall of text, consider these alternatives:
  - Progressive bullet points getting more detailed (first ones useful alone for basic cases)
  - Don't be worried to say "don't worry about this now, let's talk later."
  - Figure out the root problem instead of answering every possible interpretation
  - Declare it advanced and that you will come back later.

Ensure it can be posted quickly:

- The collaborative document gets posted to the workshop webpage. For this, it needs some minimal amount of formatting (it doesn't need to be perfect, just not horrible).
- All names and private information needs to be stripped. This is why you should rigorously tag all names with `[name=XXX]` so they can be removed (see above).
  - Learner names can be completely removed. CR staff names can be `[name=CR]` or something similar.
  - There may be other private URLs at the top or bottom.
- If possible, send the PR adding the collaborative document to the workshop webpage (though others can do this, too).

Collaborative document format example

```
# Workshop, day 1


## Lesson name
https://coderefinery.github.io/lesson/

### Episode name
https://coderefinery.github.io/01-episode/

- This is a question
  - Anwser
  - More detailed answer
- question
  - answer

### Exercises:
https://link-to-exercise/.../.../#section
20 minutes, until xx:45
Try to accomplish all of points 1-3.  Parts 4-5 are optional.

Breakout room status:
- room 2, need help with Linux permissions
- room 5, done

### Break
:::danger
We are on a 10 minute break until xx:10
:::


## Lesson 2
https://coderefinery.github.io/lesson-2/
```

Posting the collaborative document to the website

The collaborative document should be posted sooner rather than later, and hopefully the steps above will make it easy to do so quickly. You could wait a few hours, to allow any remaining questions to be asked an answered.

- Download as markdown
- Remove any private links at the top
- Adjust headings so that they are reasonable
- Look for private info and remove it
  - Search document for `[name=???]` (change to `[name=staff]` or `[name=learner]`)
  - Any names not tagged with `[name=]`
  - usernames in URLs
  - private links

Feedback template

```
## Feedback, day N

:::info
### News for day N+1
- .
- .
:::

### Today was (multi-answer):
- too fast:
- just right:
- too slow:
- too easy:
- right level:
- too advanced:
- I would recommend this course to others:
- Exercises were good:
- I would recommend today to others:
- I wouldn't recommend today:

### One good thing about today:
- ...
- ...

### One thing to be improved for next time:
- ...
- ...

### Any other comments:
- ...
- ...
```

Keypoints

- Having a collaborative document improves communication and interaction.
- Answering questions requires a dedicated person - A Collaborative Document Manager.
- The collaborative document should be posted on the web site as soon as possible.

A workshop seen from different perspectives

Objectives

- Understand the general structure of CodeRefinery workshops and why it is the way it is
- Get to know the different roles of a workshop and which ones are the most essential ones
- Understand the importance of installation instructions and how they contribute to learners success
- Understand the importance of onboarding and a welcoming community for volunteers in a workshop

Instructor note

- Teaching: 20 min
- Exercises: 15 min
- Roles journeys can be shortened to looking only at instructor role.

Note

This episode focuses on the large scale streamed workshop setup. Some of the features can also be applied to smaller scale workshops though.

Discussion

Discussion

In breakoutrooms: choose one or two questions to talk about your own experiences; summary in collaborative notes:

- What kind of roles have you been in yourself regarding workshops?
- How were you prepared for your role?
- What are the things you would like to know before a workshop?
- How are you preparing your participants for your trainings?
- What was the best workshop experience for you as learner, helper or instructor? What made it great?

One workshop - many parts

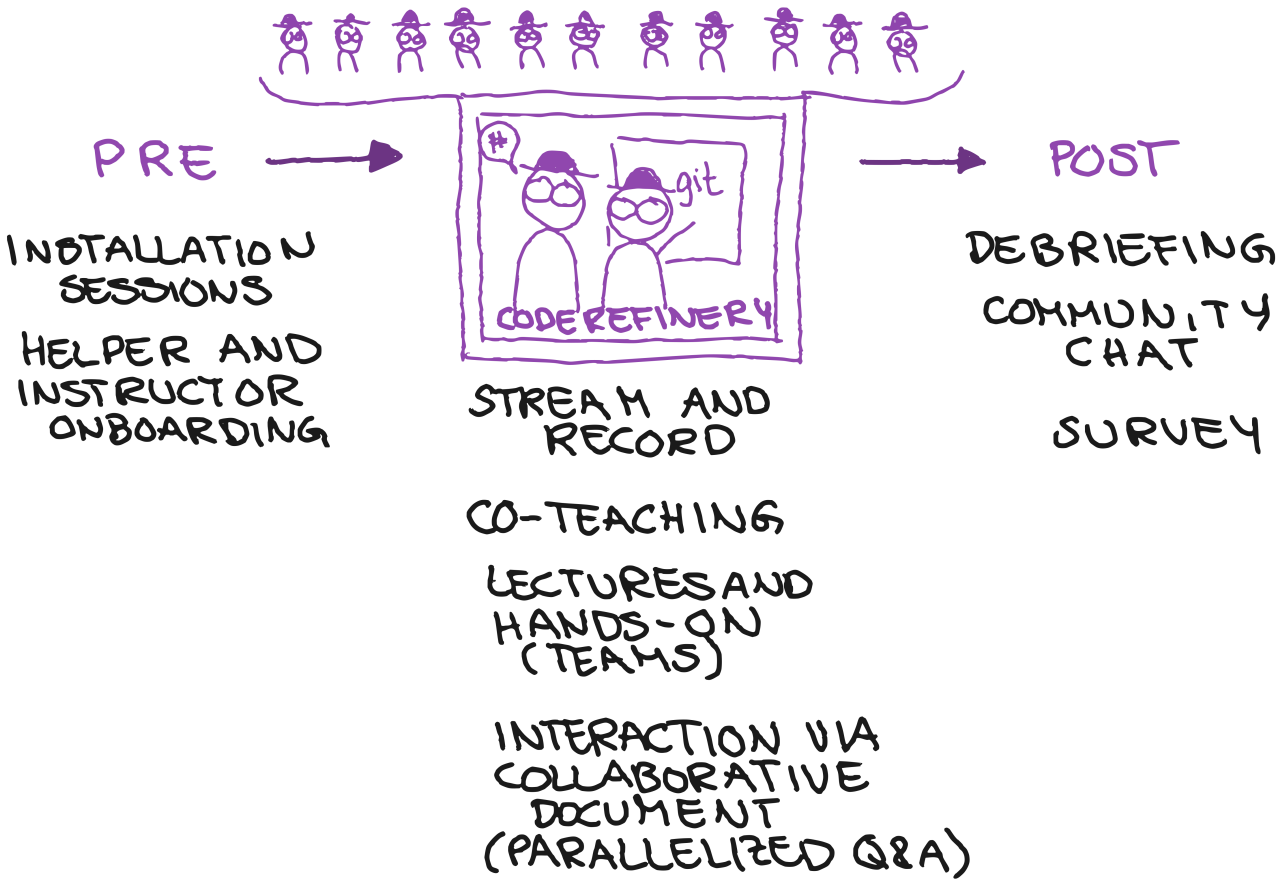


Image showing the different phases of a CodeRefinery workshop described below. Also showing that while there is only two people teaching, many people work behind the scenes.

Before the workshop

- During winter/summer: “Someone” takes the coordinator role for the next workshop
- Coordinator fills other roles:
  - Coordinator: collects necessary lesson updates, supports other coordinators
  - Registration coordinator: Sets up web page, starts and manages registration
  - Instructor coordinator: Finds instructors and onboards them
  - Advertising coordinator: Prepares advertizement texts and finds people to distribute them
  - Team coordinator: Gathers local organizers/teams
  - Bring your own code session coordinator: If available, offer BYOC session after main workshop
- Instructor: Onboarding with (instructor-) coordinator and previous instructor of lesson; update of lesson materials (urgent issues and personal touches)
- Local organizer/Team lead: Group onboarding ~ week/or two before workshop
- Learner: Gets installation instructions, invited to installation help session, workshop info from event page and summary via e-mail
- Collaborative notes manager sets up the notes document

Note

Onboarding manuals

- **Outline of what will happen during the workshop**
- Discussion of different strategies to handle the exercise sessions
- Lowering the barrier to ask for support by meeting some organizers
- Q&A

Note

Installation instructions

- Instructions for all operating systems
- Goal: Learners leave the workshop and are ready to apply the learned tools and techniques to their own work
- Support session for installation challenges

During the workshop

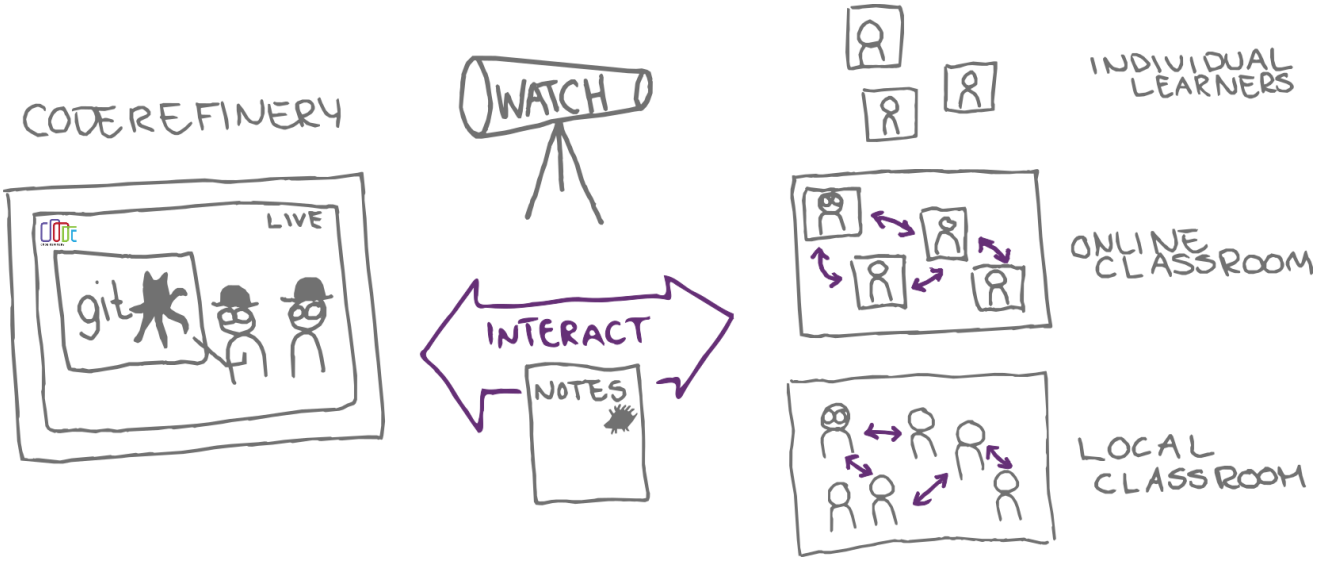


Image showing the different ways learners can do the exercises. No matter which way they choose, they always watch the stream and interact with instructors and organizers via collaborative notes. Individual learners do the exercises alone, people in online or local classrooms in addition get to discuss with their peers.

- Everyone watches stream
- Co-instructors - Present content - Answer questions in collaborative notes
- Collaborative notes manager
  - Keeps the collaborative notes clean
  - Helps answering questions
  - Adds sections
  - Archives the document after each day
- Learners do exercises individually or in team
- Local organizer / Team lead
  - Guides learners through exercises

    - Facilitates discussion
- (Registration-) coordinator sends out e-mails to participants after each day (summary + preparation)
- Director/Broadcaster manages the streaming (see session 4)

After the workshop

- Coordinator collects lessons learned based on experience and feedback and turns them into issues
- Coordinator organizes debriefing week after the workshop, where anyone can join to give feedback
- “Someone” sends out post-workshop survey half a year after workshop
- Coordinator organizes “Bring your own code” sessions
- Broadcaster prepares and shares recording (see session 4)

Note

Bring your own code sessions

We want to support learners to apply what they have learned in the workshop to their own work by offering one/two sessions where they can drop in, and discuss challenges with CodeRefinery instructors and helpers.

Image showing a classroom learning about git leading to a discussion between student and teacher to start a welcoming environment for asking for help.

This sounds like a lot of people and time investment!

- Many roles can be combined or adjusted as needed
- Some roles can be taken on by multiple people

So how many people are needed for this kind of workshop?

- Recommended minimum: 2-3 people
- Optimum: ~ 4-5 people The more people you involve, the more time goes into coordination work.

How did we get to this setup?

- **Collaborative** in-person workshops since 2016 around the Nordics
- Moved online in 2020

- Started with “traditional zoom” workshops, breakoutrooms, volunteer helpers
- Thought about scaling and ease of joining -> current setup (More about this in session 4)
- Continuous development

We also still do smaller scale local workshops, if instructors are available. **You can offer your own CodeRefinery workshop** by inviting other CodeRefinery instructors. You can also reuse single lessons and integrate them into your own teaching.

Workshop roles and their journeys

Individual learner journey

- Registration
- Installation
- (if appicable: Invited to local meetup)
- Workshop
  - Watch stream
  - Q&A in notes
  - Exercises alone or in team
  - Daily feedback
- Debrief/ Feedback session
- Post workshop survey

Instructor journey

- Indicate interest in CodeRefinery chat
- Onboarding
- Lesson material updates
- Teaching
- (if wished: supports answering questions in notes)
- Debrief

Team lead / Local host journey

- Registration
- (if applicable: run own registration)
- Onboarding
- Workshop
  - Watch stream
  - Facilitate exercises/discussions
  - Daily feedback
- Debrief / feedback / survey

Other roles

Director and broadcaster roles will be discussed in Session 4. For more in-depth descriptions and other roles (host, instructor, registration coordinator, etc), see also our CodeRefinery manuals.

Different roles as stepping stones for community involvement

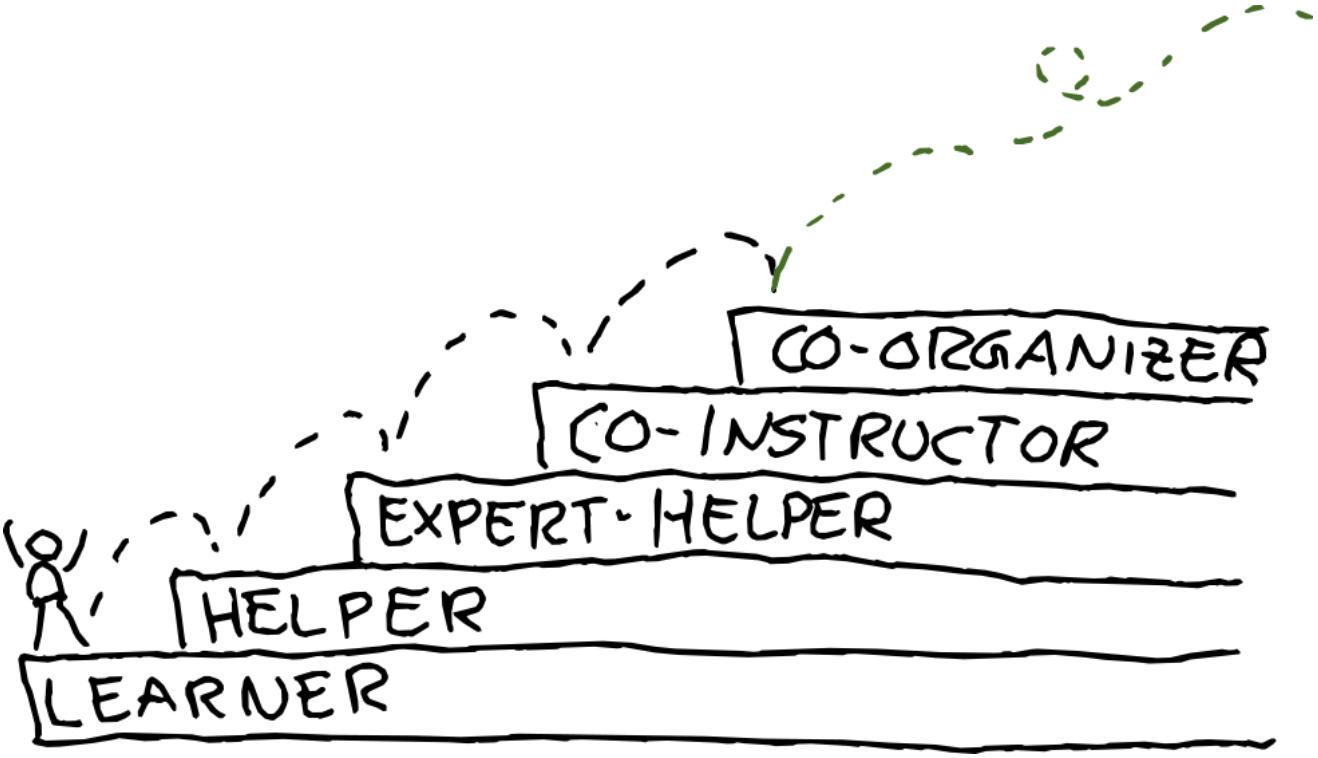


Image showing a person jumping from step to step following learner, helper, expert helper, co-instructor, co-organizer.

There is no "one way" to get involved. Do what feels right. Interested in teaching with us? Let us know and we can discuss what is a suitable path **for you**!

Keypoints

- CodeRefinery workshops are a collaborative effort with many different roles
- Onboarding the different roles is one key aspect of our workshops
- Everyone has their own path

Sound

Objectives

- Understand that sound quality is very important
- Evaluate your and others sound quality and know how to improve it
- Test tips and tricks for achieving good sound quality

Instructor note

- Teaching: 10 min
- Exercises: 10 min

The importance of audio

- Pleasing audio quality makes events much more enjoyable
- Audio is one of the most important things you can control
- Things that can go wrong:
  - Too quiet
  - Too loud
  - Instructors' volumes imbalanced
  - Background noise
  - Low-quality/breaking up audio hard to hear.
  - "Ducking" (first words lower volume, or lower volume other times)

Tips for good sound quality

- Have a headset with mounted microphone
  - Even if you have a professional external microphone, it doesn't matter if your room has bad acoustics.
  - The close pickup can't be beat with normal tools.
  - As long as it's headset mounted, price doesn't seem to matter *that* much.
- Don't use Bluetooth
  - Bluetooth can have too much latency (300-500ms)
  - This may seem small but for interactive work, it's a lot
  - Use a wired headset, or wireless with a non-Bluetooth USB plug (like gaming headsets have). These have much lower latency.
  - Bluetooth 5 can have much lower latency, but you probably shouldn't count on that without testing.
  - It can also have lower sound quality on some devices due to bandwidth limitations.
- Once you have a headset, turn input noise cancellation to low (wherever it might be: headphone, meeting software, etc.).

Balancing and dynamic adjustment

- It's important that instructors volumes match.
- An exercise will go over a systematic procedure for matching volumes.
  - Practice so that you can do this quickly.
- May need re-adjusting when instructors swap out or start getting excited.
- An exercise below will demonstrate our procedure.

Speak up when there are problems

- If you notice someone with audio issues, let them know right away (voice if bad, or chat/notes if less urgent).
- Take the time to fix it as soon as practical.
- Make a culture of *speaking up, helping, and not suffering*.

Recommendations

- Procure some reasonable headset.
- Low/medium-priced gaming-type headsets have worked well for us.
  - (gaming headsets usually aren't Bluetooth, because gaming needs low latency.)
- Show this page to your workplace (if you have one) and call a good headset work equipment.

Exercises

✍ Sound-1: Evaluate sound quality

It's important to be able to discuss with others the quality of their audio. They can't hear themselves, after all.

- Within the teams, discuss each person's audio quality and what kind of setup they have.
- Be respectful and constructive (and realize that people came prepared to listen, not teach).
- Consider, for example
  - Volume
  - Clarity
  - Background noise
  - Noise cancellation artifacts
  - "Ducking": first words at lower volume or missing
- Discuss how to bring this up during other courses and meetings.

✍ Sound-2: Adjust volume up and down

In addition to individual quality, it is important that sound is balanced among all instructors. Have you ever been to a event when one person is too quiet and they can't make themselves any louder? And they can't adjust it?

You should ensure that you have a range of volumes available to you. You might have to look into several different locations in order to make the full adjustment ().

- Go to your sound settings
- One by one, each person
  - Adjusts their microphone volume so quiet that they can't be heard.
  - Adjusts their microphone volume so that it is extremely loud (this may require going beyond 100% if possible).
- Basically, make sure you aren't so close to either end that you have no potential to make an adjustment in that direction.
- Everyone tries to set the volume to something reasonable, in preparation for the next exercise.

Once you know where these settings are , you won't be panicked when the volume is too low or high during a course.

✍ Sound-3: Do a balance check

It's important that instructor audio is balanced (the same volume). But how can you do this?

- Pick a leader.
- The leader decides the order ("I am first, then [name-1] and [name-2]")
- The leader says "one". Everyone else says "one" in the order specified.
- The leader says "two". Everyone else says "two" in the order.
- The leader asks for opinions on who they think is louder or softer. If there are more than three people, you can figure it out yourselves. With less than two, you have to ask someone in the audience.

Example:

- Leader: Let's do a sound check. I am first, then AAA and BBB.
- Leader: One
- AAA: One
- BBB: One
- Leader: Two
- AAA: Two
- BBB: Two
- Leader: Three
- AAA: Three
- BBB: Three
- Leader: How did that sound to everyone?
- [Someone else]: Leader and BBB were pretty similar but AAA is a bit lower.

Summary

❗ Keypoints

- Audio quality is important and one of the most notable parts of the workshop.
- Improving audio isn't hard or expensive, but does require preparation.

How to prepare a quality screen-share

❗ Objectives

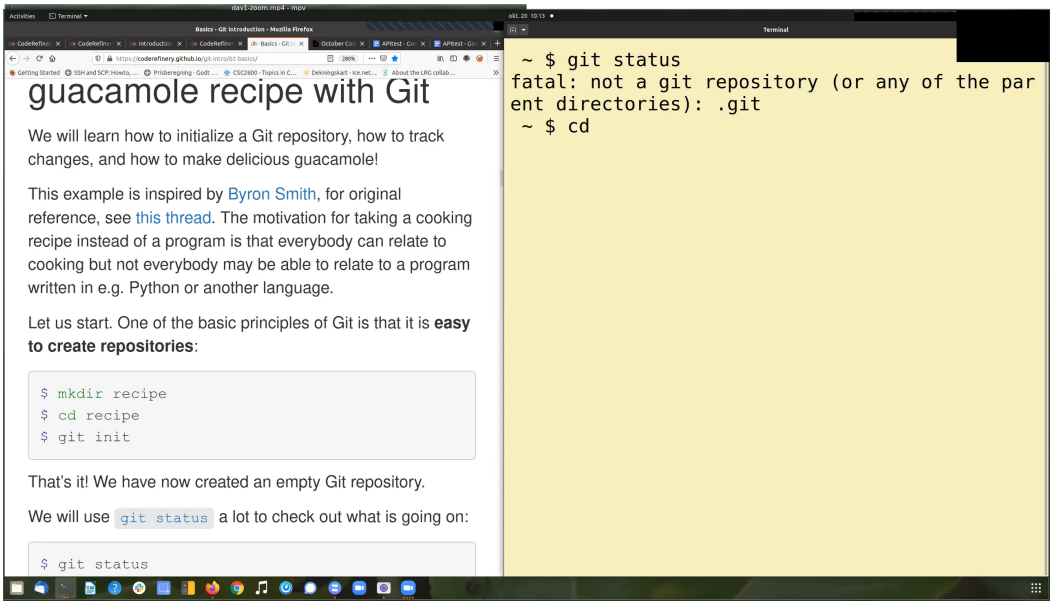
- Discuss the importance of a well planned screen-share.
- Learn how to prepare and how to test your screen-share setup.
- Know about typical pitfalls and habits to avoid.

Instructor note

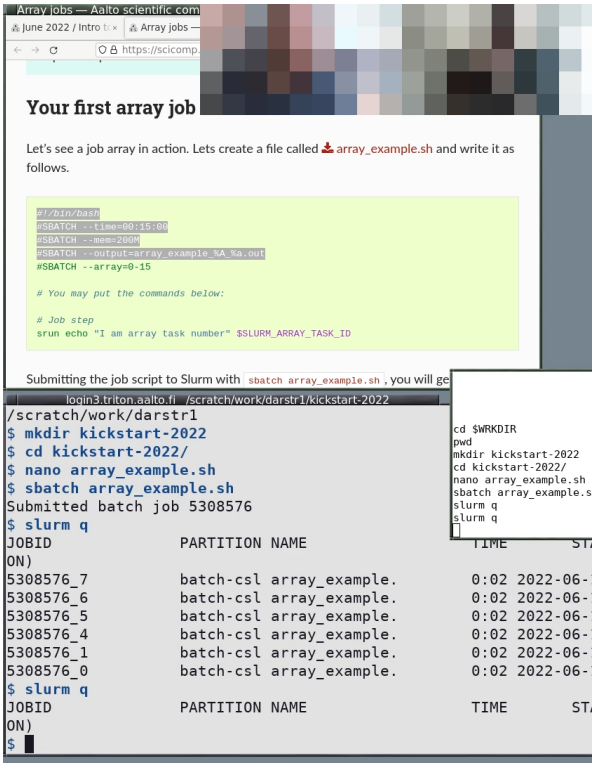
- Discussion: 15 min
- Exercises: 15 min

Share portrait layout instead of sharing entire screen when teaching online

- Many learners will have a smaller screen than you.
- You should plan for learners with **only one small screen**.
- A learner will **need to focus on both your screen share and their work**.
- Share a **portrait/vertical half of your screen** (840 × 1080 is our standard and your maximum).
- Zoom provides a “Share a part of screen” that is good for this.
  - Our latest streaming setup (day 4) can take the portrait part for you so you can share landscape.
  - Zoom + Linux + Wayland display manager doesn’t have “Share a portion of the screen”
    - You can share a single window portrait.
    - Or you can start your desktop session in “X11” or “Xorg” legacy mode.
    - Or possibly other workarounds (does anyone know other solutions?)



A FullHD 1920x1080 screen shared. Learners have to make this really small if they want to have something else open.

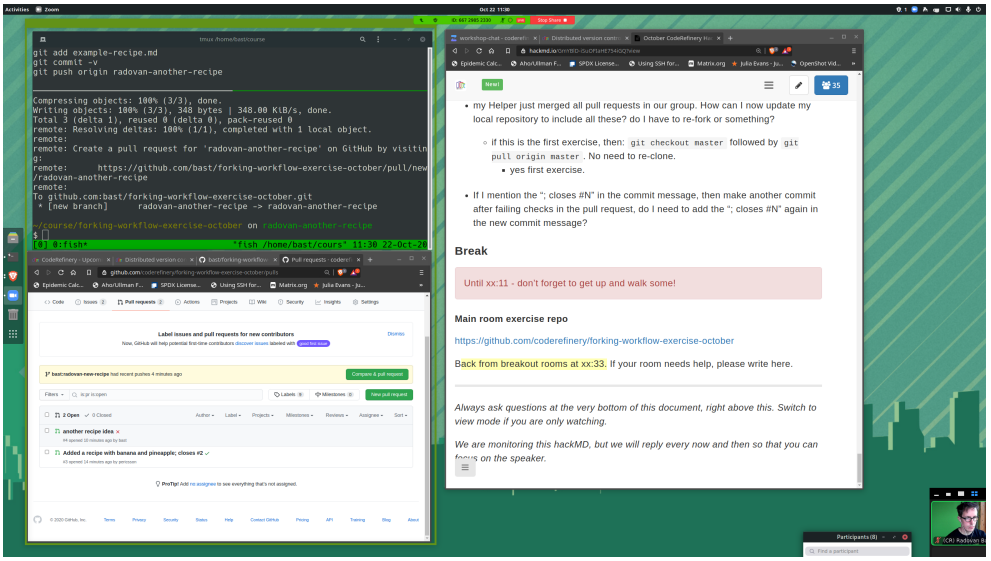


Portrait layout. Allows learners to have something else open in the other half.

Motivation for portrait layout:

- This makes it easier for you to look at some other notes or to coordinate with other instructors at the same time without distracting with too much information.
- This makes it possible for participants to have something else open in the other screen half: terminal or browser or notebook.

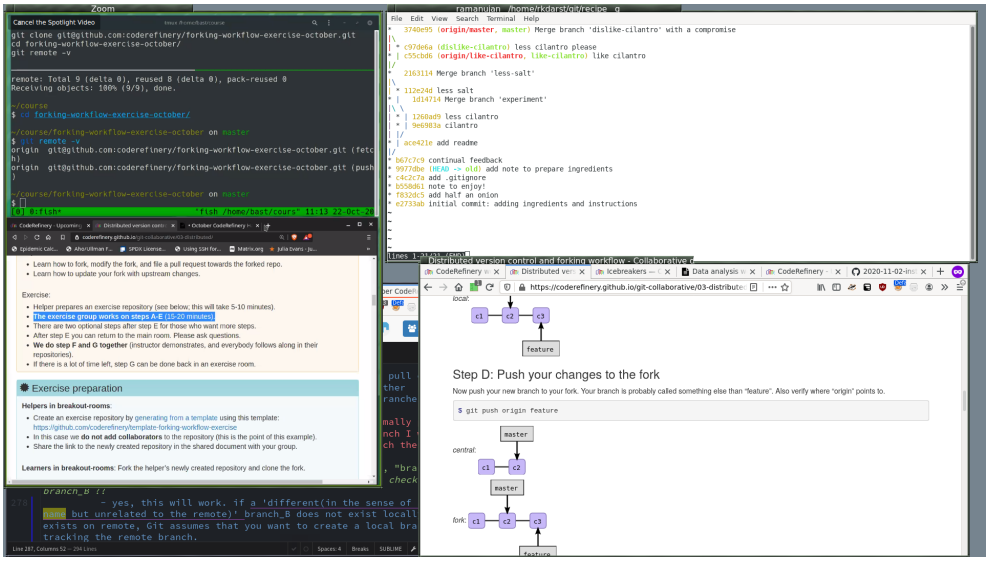
Instructor perspective



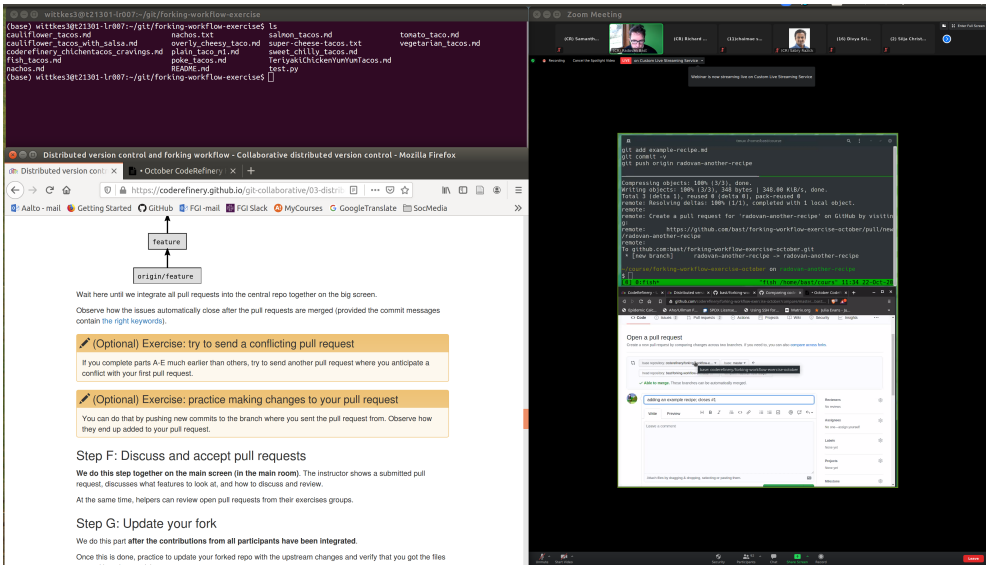
***I1**: This is how it can look for the instructor. Zoom is sharing a portion of the left side, the right side is free for following notes, chat, etc.*

Learner perspective

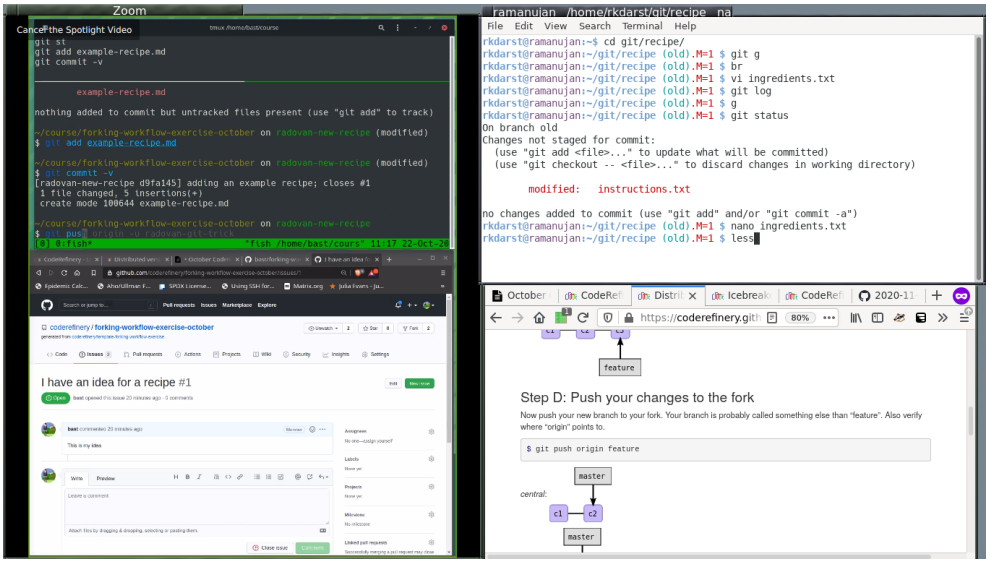
Here are three examples of how it can look for the learner.



***L1**: Learner with a large screen, Zoom in dual-monitor mode so that the instructor pictures are not shown. Screen-share is on the left side, collaborative notes at bottom left, terminal and web browser on the right.*



***L2**: A learner with a single large screen (Zoom in "single monitor mode"). Instructor screen share at right, learner stuff at left.*



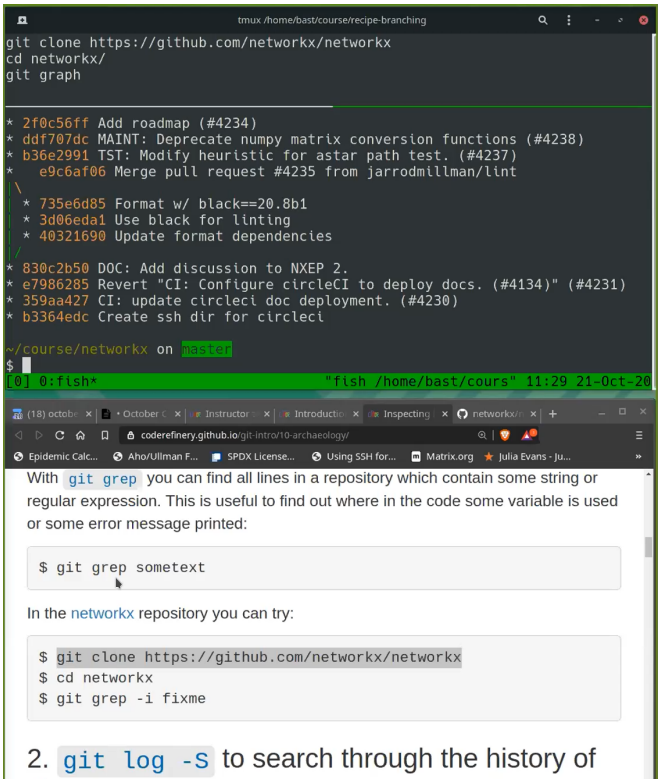
***L3**: A learner with a particularly small screen. Instructor screen-share at left, your windows at right.*

Share the history of your commands

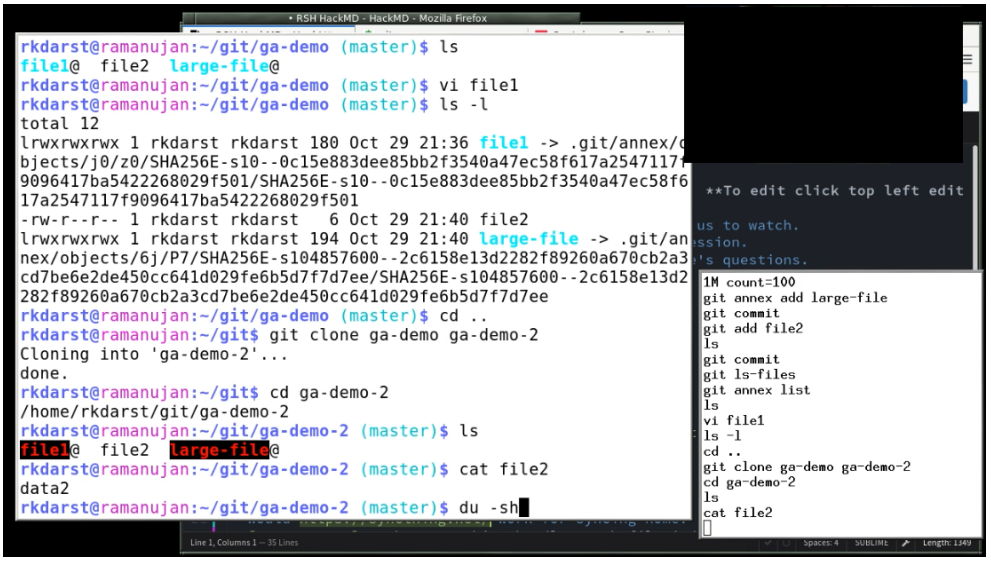
Even if you type slowly it is almost impossible to follow every command. We all get distracted or want to read up on something or while following a command fails on the learner laptop while instructor gets no error.

Add pauses and **share the commands that you have typed** so that one can catch up.

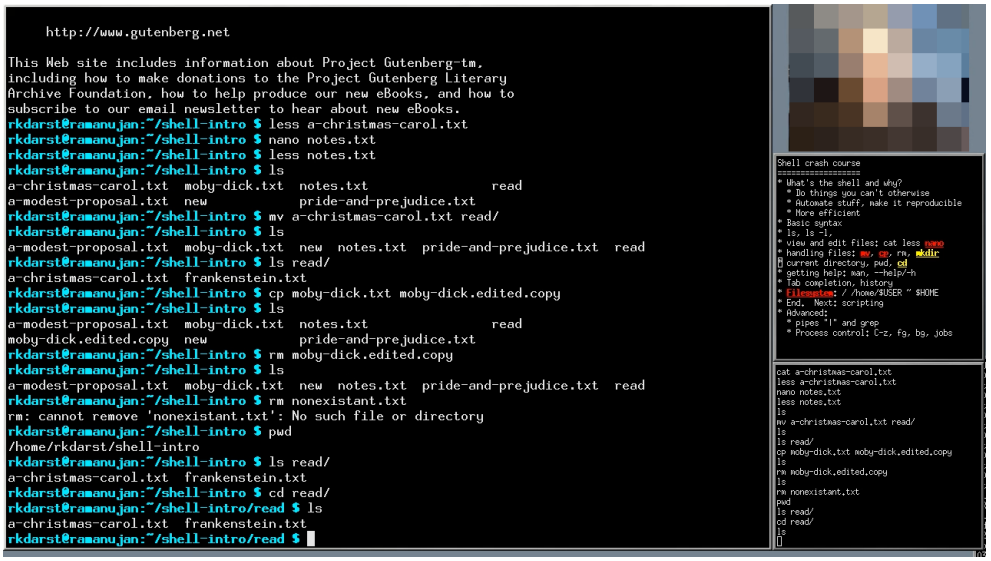
Below are some examples (some more successful than others) of sharing history of commands.



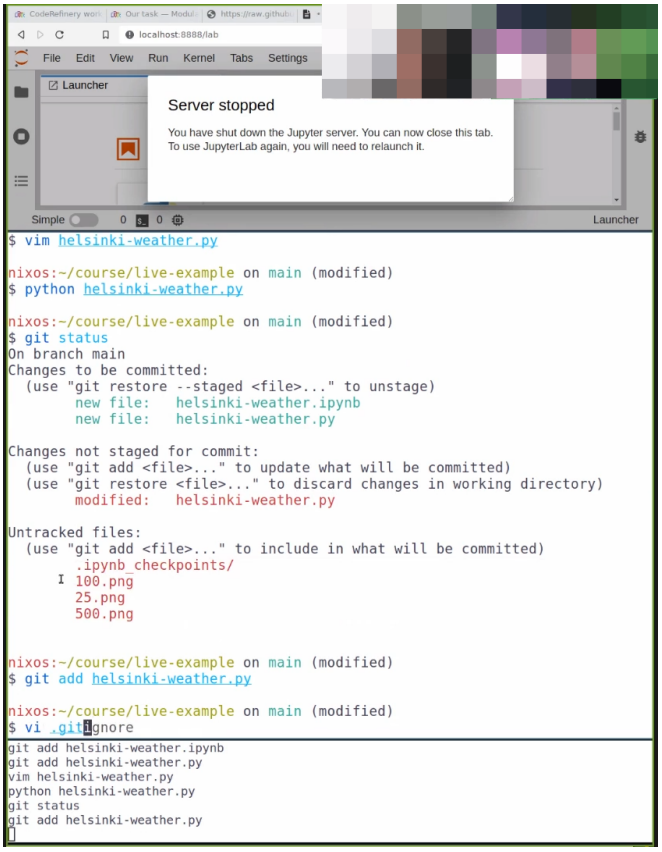
***H1**: A vertical screen layout shared. Note the extra shell history at the top. The web browser is at the bottom, because the Zoom toolbar can cover the bottom bit.*



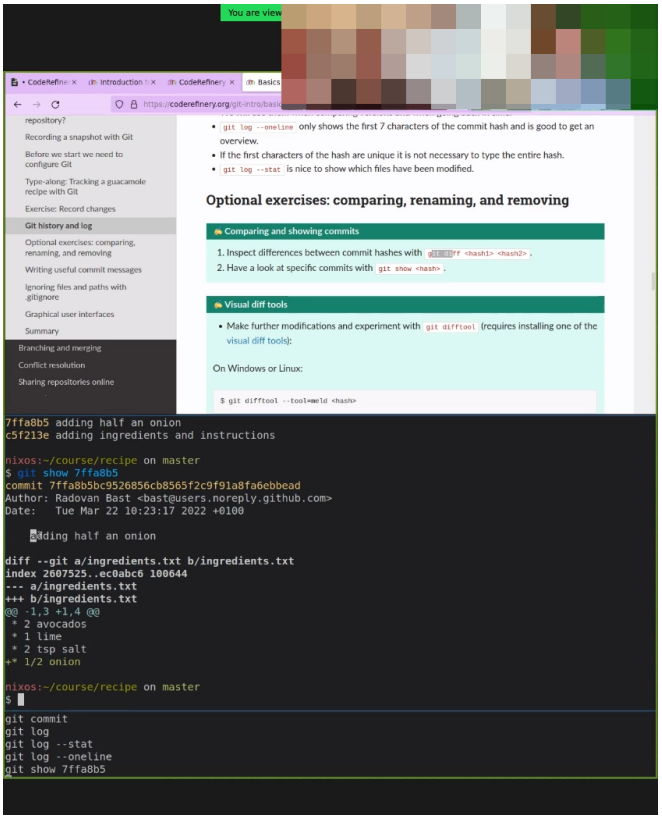
***H2**: This isn't a screen-share from CodeRefinery, but may be instructive. Note the horizontal layout and shell history at the bottom right.*



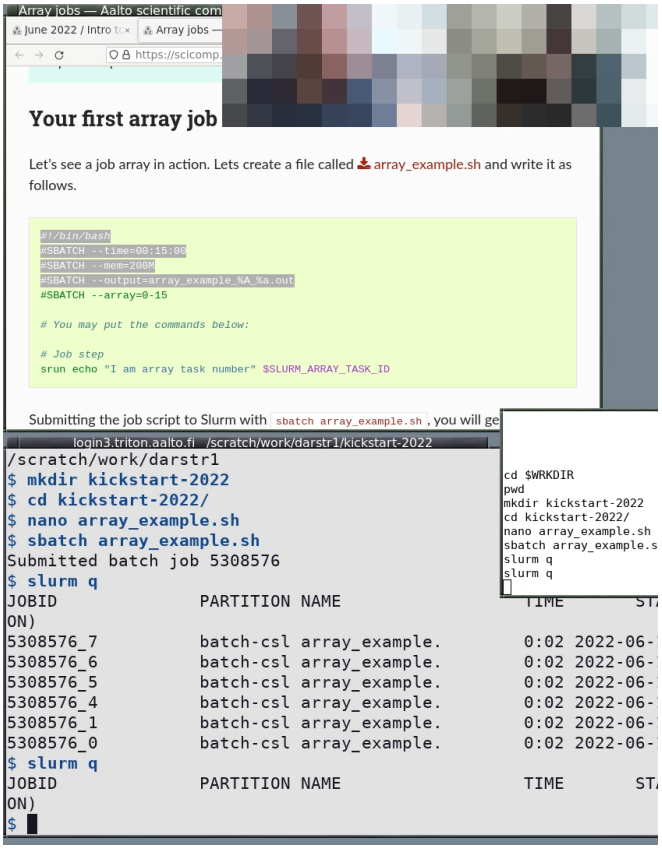
H3: Similar to above, but dark. Includes contents on the right.



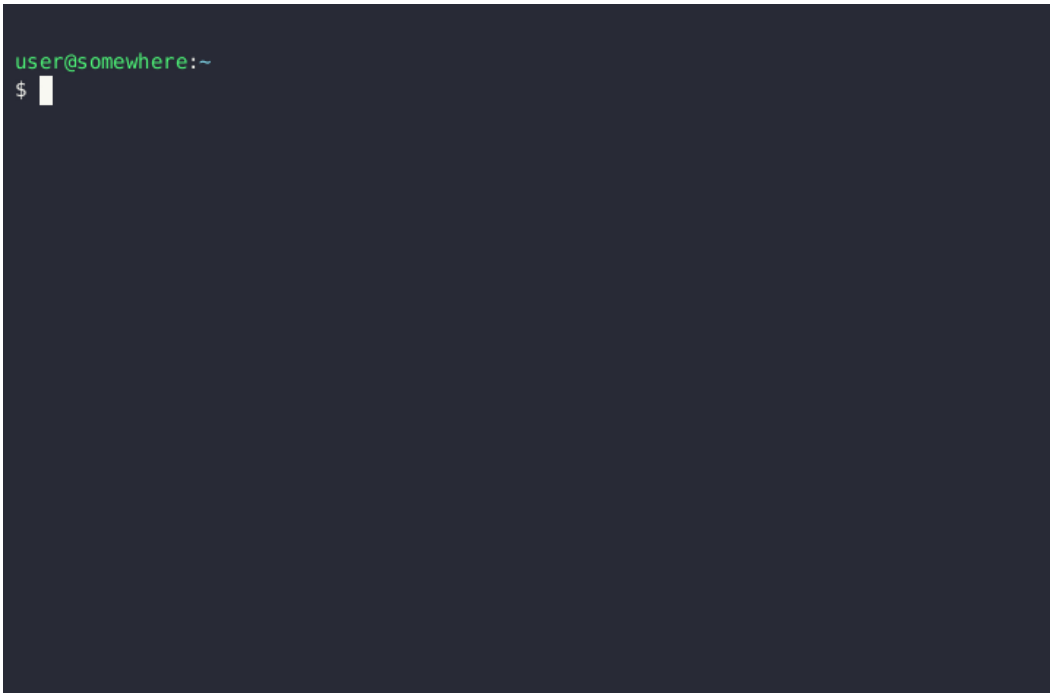
H4: Jupyter + terminal, including the `fish` shell and the terminal history.



H5: *Lesson + terminal,* `tmux` *plus terminal history and dark background.*



H6: *HPC Kickstart course. Note the colors contrast of the windows and colors of the prompt and text. The history is smaller and doesn't take up primary working space. The working directory is in the window title bar.*



H7: Show command history "picture-in-picture", in the same terminal window.

How to configure history sharing

You need to find a way to show the recent commands you have entered, so that learners can see the recent commands. Below are many solutions. Try them out and see what works for you.

- prompt-log: It adds a interesting idea that the command you enter is in color and also provides terminal history before the command returns.
- **Simple**: The simple way is `PROMPT_COMMAND="history -a"` and then `tail -f -n0 ~/.bash_history`, but this doesn't capture ssh, sub-shells, and only shows the command after it is completed.
- **Better yet still simple**: Many Software Carpentry instructors use this script, which sets the prompt, splits the terminal window using tmux and displays command history in the upper panel. Requirement: tmux
- **Better (bash)**: This prints the output before the command is run, instead of after. Tail with `tail -f ~/demos.out`.

```
BASH_LOG=~/demos.out
bash_log_commands () {
    # https://superuser.com/questions/175799
    [ -n "$COMP_LINE" ] && return  # do nothing if completing
    [[ "$PROMPT_COMMAND" =~ "$BASH_COMMAND" ]] && return # don't cause a preexec
for $PROMPT_COMMAND
    local this_command=`HISTTIMEFORMAT= history 1 | sed -e "s/^[ ]*[0-9]*[ ]*//"`;
    echo "$this_command" >> "$BASH_LOG"
}
trap 'bash_log_commands' DEBUG
```

- **Better (zsh)**: This works like above, with zsh. Tail with `tail -f ~/demos.out`.

```
preexec() { echo $1 >> ~/demos.out }
```

- **Better (fish)**: This works like above, but for fish. Tail with `tail -f ~/demos.out`.

```
function cmd_log --on-event fish_preexec ; echo "$argv" >> ~/demos.out  ; end
```

- **Better (tmuxp)**: This will save some typing. TmuxP is a Python program (`pip install tmuxp`) that gives you programmable `tmux` sessions. One configuration that works (in this case for `fish` shell):

```
session_name: demo
windows:
  - window_name: demo
    layout: main-horizontal
    options:
      main-pane-height: 7
    panes:
      - shell_command:
          - touch /tmp/demo.history
          - tail -f /tmp/demo.history
      - shell_command:
          - function cmd_log --on-event fish_preexec ; echo "$argv" >>
/tmp/demo.history  ; end
```

- **Windows PowerShell**: In Windows Terminal, a split can be made by pressing `CTRL+SHIFT+=`. Then, in one of the splits, the following PowerShell command will start tracking the shell history:

```
Get-Content (Get-PSReadlineOption).HistorySavePath -Wait
```

  Unfortunately, this only shows commands after they have been executed.

- Tavatar: shell history mirroring teaching tool can copy recent history to a remote server.
- history-window: Show command history "picture-in-picture" when teaching command line. Requires Bash.

Font, colors, and prompt

Terminal color schemes

- Dark text on light background, *not* dark theme. Research and our experience says that dark-text-on-light is better in some cases and similar in others.
- You might want to make the background light grey, to avoid over-saturating people's eyes and provide some contrast to the pure white web browser. (this was an accessibility recommendation when looking for ideal color schemes)
- Do you have any yellows or reds in your prompt or program outputs? Adjust colors if possible.

Font size

- Font should be large (a separate history terminal can have a smaller font).
- Be prepared to resize the terminal and font as needed. Find out the keyboard shortcuts to do this since you will need it.

Prompt

At the beginning of the workshop your goal is to have a shell **as easy to follow as possible** and **as close to what learners will see on their screens**:

- Your prompt should be minimal: few distractions, and not take up many columns of text.
- prompt-log does this for you.
- The minimum to do is is `export PS1='\$ '`.
- Blank line between entries: `export PS1='\n\$ '`.
- Have a space after the `$` or `%` or whatever prompt character you use.
- Strongly consider the Bash shell. This is what most new people will use, and Bash will be less confusing to them.
- Eliminate menu bars and any other decoration that uses valuable screen space.
- Add colors only if it simplifies the reading of the prompt.

Later in the workshop or in more advanced lessons:

- Using other shells and being more adventurous is OK - learners will know what is essential to the terminal and what is extra for your environment.

Try to find a good balance between:

- Showing a simple setup and showing a more realistic setup.
- Showing a consistent setup among all instructors and showing a variety of setups.

Habits we need to un-learn

- **Do not clear the terminal**. Un-learn CTRL-L or `clear` if possible. More people will wonder what just got lost than are helped by seeing a blank screen. Push `ENTER` a few times instead to add some white space.
- **Do not rapidly switch between windows** or navigate quickly between multiple terminals, browser tabs, etc. This is useful during your own work when nobody is watching, but it is very hard to follow for learners.
- Avoid using **aliases** or **shortcuts** that are not part of the standard setup. Learners probably don't have them, and they will fail if they try to follow your typing. Consider even to rename corresponding files (`.bashrc` , `.gitconfig` , `.ssh/config` , `.ssh/authorized_keys` , `.conda/*`).
- Be careful about using **tab completion** or **reverse history search** if these haven't been introduced yet.

Desktop environment and browser

- Try to remove window title bars if they take up lots of space without adding value to the learner.
- Can you easily resize your windows for adjusting during teaching?
- Does your web browser have a way to reduce its menu bars and other decoration size?
  - Firefox-based browsers: go to `about:config` and set `layout.css.devPixelsPerPx` to a value slightly smaller than one, like `0.75`. Be careful you don't set it too small or large since it might be hard to recover! When you set it to something smaller than 1, all window decorations become smaller, and you compensate by zooming in on the website more (you can set the default zoom to be greater than 100% to compensate). Overall, you get more information and less distraction.

How to switch between teaching setup and work setup?

- Make a dedicated "demos" profile in your terminal emulator, if relevant. Or use a different terminal emulator just for demos.
- Same idea for the browser: Consider using a different browser profile for teaching/demos.
- Another idea is to containerize the setup for teaching. We might demonstrate this during the Sharing teaching gems session later.

❗ Keypoints

- Share **portrait layout** instead of sharing entire screen
- **Adjust your prompt** to make commands easy to read
- **Readability** and beauty is important: adjust your setup for your audience
- **Share the history** of your commands
- Get set up a **few days in advance** and get feedback from someone else. Feedback and time to improve is very important to make things clear and accessible. 10 minutes before the session starts is typically too late.

Exercises

✍ Evaluate screen captures (20 min)

Evaluate screenshots on this page. Discuss the trade-offs of each one. Which one do you prefer? Which are useful in each situation?

Please take notes in the collaborative document.

✍ Set up your own environment (20 min)

Set up your screen to teach something. Get some feedback from another learner or your exercise group.

Other resources

- https://coderefinery.github.io/manuals/instructor-tech-setup/
- https://coderefinery.github.io/manuals/instructor-tech-online/

Computational thinking

Objectives

- Explain what is computational thinking
- Get to know how the theory of computational thinking can be used in teaching
- Short exercise

Instructor note

- Teaching: 20 min
- Exercises: 10 min

Materials available as slides.

Keypoints

- Computational Thinking consists of 4 main parts: decomposition, pattern recognition, abstraction and algorithmic design.
- How can this be a useful framework for solving problems?
- How can this be used practically?

CodeRefinery teaching philosophies

Objectives

- Get to know the teaching philosophies of CodeRefinery instructors

Instructor note

- Teaching: 10 min
- Discussion: 20 min

Introduction

During this episode we split into breakoutrooms and discuss own teaching philosophies. Collect your teaching philosophies in the collaborative document. We will be going through these and the end of the lesson.

✍ Ice-breaker in groups (20 minutes)

- Share your approach to teaching and your teaching philosophy with your group.
- Please share your tricks and solutions in the live document for others.

Additional ice-breaker questions:

- What is your motivation for taking this training?
- How structured or informal are your own teaching needs?
- What difference do you notice between the teaching what we (also Carpentries) do and traditional academic teaching?
- What other skills need to be taught, but academic teaching isn't the right setting?

Instructor views

Here CodeRefinery instructors share their training philosophy to show that we all have different teaching styles and how these differences are beneficial to CodeRefinery workshops.

It is important to explain how much we value individuals and that there is not one way to teach but as many ways as individuals. We want to help each other to find the way that is best for each of us.

⚙ Video recordings

We have recorded some of the below as videos: https://www.youtube.com/playlist?list=PLpLblYHCzJAAHF89P-GCjEXWC8CF-7nhX

✍ Bjørn Lindi

My teaching style has changed a bit since I started with CodeRefinery. In the beginning I had this "BLOB" (Binary Large OBject) of knowledge and experience that I wanted to to convey to the participants. With experience and some help from the Carpentries Instructor training, I have realized I need to serialize the "BLOB", to be able to share it with others.

In a similar fashion as you would do with a binary large object which you intend to send over the wire, you will need stop signals, check-sums and re-transmissions, when you give a lecture. I have come to appreciate the natural periods/breaks the lessons offers, the

questions raised, the errors that appear during type-along and the re-transmission. Co-instructors are good to use for re-transmission or broadening a specific topic.

When I started with CodeRefinery my inclination was to give a lecture. Today I am trying to be a guide during a learning experience, a learning experience which includes me as well. That may sound a bit self-centric, but is in fact the opposite, as I have to be more sensitive to what is going on in the room. The more conscious I am of being a guide, the better lesson.

Tools that I find useful in preparing a lesson is concept maps and Learner Personas (though I have developed too few of them):

- Concept Maps
- Learner Personas

✍ Radovan Bast

My teaching changed by 180 degrees after taking the Carpentries instructor training. Before that I used slides, 45 minute lecture blocks, and separate exercise sessions. After the Carpentries instructor training I embraced the interaction, exercises, demos, and typos.

My goal for a lesson is to spark curiosity to try things after the lesson, both for the novices ("This looks like a useful tool, I want to try using it after the workshop.") and the more experienced participants ("Aha - I did not know you could do this. I wonder whether I can make it work with X."). I like to start lessons with a question because this makes participants look up from their browsers.

Keeping both the novices and the experts engaged during a lesson can be difficult and offering additional exercises seems to be a good compromise.

For me it is a good sign if there are many questions. I like to encourage questions by asking questions to the participants. But I also try not to go into a rabbit hole when I get a question where only experts will appreciate the answer.

I try to avoid jargon and "war stories" from the professional developers' perspective or the business world. Most researchers may not relate to them. For examples I always try to use the research context. Avoid "customer", "production", also a lot of Agile jargon is hard to relate to.

Less and clear is better than more and unclear. Simple examples are better than complicated examples. Almost never I have felt or got the feedback that something was too simple. I am repeating in my head to not use the words "simply", "just", "easy". If participants take home one or two points from a lesson, that's for me success.

I prepare for the lesson by reading the instructor guide and all issues and open pull requests. I might not be able to solve issues, but I don't want to be surprised by known issues. I learn the material to a point where I know precisely what comes next and am never surprised by the next episode or slide. This allows me to skip and distill the essence and not read bullet point by bullet point.

I try to never deviate from the script and if I do, be very explicit about it.

A great exercise I can recommend is to watch a tutorial on a new programming language/tool you have never used. It can feel very overwhelming and fast to get all these new concepts and tools thrown at self. This can prepare me for how a participant might feel listening to me.

I very much like the co-teaching approach where the other person helps detecting when I go too fast or become too confusing. I like when two instructors complement each other during a lesson.

✍ Sabry Razick

My approach is to show it is fun to demystify concepts. Once a concept is not a mystery anymore, the learners will understand is what it means, where it is coming from, why it is in place and what it could it offer for their future. I try to relate concepts to real life with a twist of humour whenever possible if the outcome is certain not be offensive to any one. I use diagrams whenever possible, I have spent weeks creating diagrams that is sometime three or four sentences. That effort I consider worthwhile as my intention is not to teach, but to demystify. Once that is achieved, learners will learn the nitty gritty on their own easily and with confidence, when they have the use-case.

✍ Richard Darst

Like many people, I've often been teaching, but rarely a teacher. I tend to teach like I am doing an informal mentorship. I've realized long ago that my most important lessons weren't learned in classes, but by a combination of seeing things done by friends and independent study after that. I've realized that teaching (the things I teach) is trying to correct these differences in backgrounds.

My main job is supporting computing infrastructure, so my teaching is very grounded in real-world problems. I'm often start at the very basics, because this is what I see missing most often.

When teaching, I like lots of audience questions and don't mind going off-script a bit (even though I know it should be minimized). I find that sufficient audience interest allows any lesson to be a success - you don't have to know everything perfectly, just show how you'd approach a problem.

✍ Stepas Toliautas

I aim for my learners to understand things (concepts, techniques…), instead of just memorizing them. The phrase I use from time to time when teaching information technology topics (be it hardware or software) is “there is no magic”: everything can be explained if necessary. While CodeRefinery also emphasizes usefulness/ immediate applicability of the content, often having the correct notions actually helps to start using them sooner and with less mistakes.

I try to guide my audience through the presented material with occasional questions related to previous topics and common knowledge – to help them link the concepts already during the lesson. I’m also fully aware that waiting for someone to answer is quite uncomfortable in-person and doubly so in an online environment, especially if no one uses their cameras :)

And if I get the question I don’t have the answer for (or material to show it) at the moment, in university classes I prepare and come back to it next time. While with one-off workshops there might not be a “next time”, open-source material used by CodeRefinery allows to have the same outcome: the latest and greatest version stays available for self-study afterwards.

Summary

Keypoints

- People have different viewpoints on teaching.

Co-teaching

Objectives

- Get to know the principle of co-teaching: How we do it and how you can too.
- Learn the team teaching concept and how to tailor it to your situation.

Instructor note

- Teaching: 15 min
- Exercises: 10 min
- Discussion: 5 min

Overview

CodeRefinery lessons benefit from the application of the concepts of **co-teaching**.

Co-teaching

> Co-teaching can be defined as “the division of labor between educators to plan, organize, instruct and make assessments on the same group of students, generally in a common classroom, and often with a strong focus on those teaching as a team complementing one another’s particular skills or other strengths”.

Co-teaching can be used in various forms, some of which are present in our workshops:

- **Teaching + support**, e.g. one of the teachers leading instruction while the other watches over and maintains the collaborative document (HackMD/HedgeDoc/...).
- Another similar example is **remote learning groups** that watch the streamed CodeRefinery lessons guided by the local instructors.
- Having open-source material and planning jointly allows **multiple instances** of a lesson to be held by multiple teachers:
  - *parallel teaching*, to different audiences at the same time,
  - *alternative teaching*, to different audiences at the same or different time, with potential content adjustments (for example, different installation procedures).
- **Team teaching**, where the lesson is presented by multiple (in most cases, two) teachers who actively engage in conversation with each other. The team-teaching concept is explained in more detail in the CodeRefinery manual.

In reality, different forms are very often mixed or fused together, even within a single lesson.

Co-teaching is not an online-only concept. However, it is very practical in online teaching due to larger number of instructors and learners potentially available to join a teaching session.

Co-teaching and team teaching benefits

- It **saves preparation time**. Co-teachers can rely on each other’s strengths while creating/revising the material as well as in unexpected situations during the lesson.
- It **helps with onboarding new instructors**. One of the co-teachers can be learning at the same time, either the subtleties of the material taught (in this case literally being the “voice of the audience”) or the teaching process itself.
- Team teaching **looks more interactive and engaging** to the audience in many cases, without forcing the learners to speak up if they can’t or don’t want to do so.
- It also **ensures responsive feedback and less workload** by having more active minds.

Are there any downsides?

Not every learner and not every instructor might like the team-teaching approach.

- It might seem **less structured**, unprepared, and chaotic, even with preparation.
  - It might create situations where instructors accidentally talk over each other or “interrupt” and change the flow of the lesson.

    - For some instructors it can be stressful to not know in advance what questions they get asked from the co-instructor.
    - Sometimes when an unexpected question is asked that throws the other instructor off, it can add to the feeling of chaos and unpreparedness.
- It can be interactive and engaging but it can also end up awkward if the co-teachers don't have a good synergy.
    - Can sound awkward: Main instructor talking all the time and at the end asking co-instructor whether everything is clear and co-instructor only saying "yes".
    - Possibly more engaging: Co-instructor asking questions which help with the flow and a common understanding of the material.

Team teaching specifics

- For successful team teaching, additional **coordination** is needed, first of all to agree on the teaching model (see below) and the person in control (the **director**) for the lesson or its parts.
- It's useful to keep track of the **lecture plan**. The discussion is a good way to make lesson more interactive and adjust to the audience, but deviating too much will become disorienting (for example, if someone dropped their attention for a minute and now is trying to catch-up by reading lecture notes).
- Experienced solo teacher might have a habit to keep talking (lecturing), while the co-teacher might not want to "interrupt". Therefore, it is important for the leading presenter to anticipate and **allow for remarks/ questions**, and this can be different from one's previous teaching style at first.

Team teaching models

We propose two basic models, but of course there is a constant continuum.

Guide and demo-giver

One person serves the role of **guide**, explaining the big picture and context of the examples.

Another, the **demo-giver**,

- shows the typing and does the examples,
- might take the role of a learner who is asking about what is going on, to actually explain the details, or to comment occasionally.

Hands-on demos and exercises work especially well like this.

Presenter and interviewer

In this case, one is the **presenter** who is mostly explaining (including demos or examples), and trying to move the forward through the material.

Another, the **interviewer**,

- serves as a learner or spotter,
- fills in gaps by asking relevant questions,
- tries to comment to the presenter when things are going off track.

This can be seen as closer to classical teaching, but with a dedicated and prepared “voice of the audience”.

Exercise

✍ Discuss the models of team teaching (10 min)

While in breakout rooms, discuss one of the basic team-teaching models presented here:

- Have you already tried this or similar model in your teaching?
- Does it seem natural to apply this model in your subject area (tell what it is)? How could it be adapted to fit best?

Write your comments in the collaborative document.

Summary

Keypoints

- Co-teaching focuses on complementing individual skills and strengths in teaching process.
- Co-teaching may save time, reduce teachers’ workload and make lessons more interactive/ engaging.
- Team teaching requires some adjustments in lesson preparation and delivery.

Sharing teaching gems

Objectives

- Our goal is to share our teaching tricks and tools and demonstrate them in very short presentations/discussions.

Instructor note

- Demonstrations: 35 min

Here we encourage participants to take the screen share for 3 minutes and demonstrate either some tool or trick they found useful for their teaching or something from a course you liked.

Session 4 intro

This session covers streaming and technical production. Some introductory notes:

- (As of 2024) This is the first and only comprehensive introduction to our online streaming.
- These practices are new and well-refined internally, but a *different* kind of refinement is needed to teach and reuse them.
- The lessons have outlines of what to talk about, but it's just an outline. It is *not* refined, since thees things are *new*. Many things will have to be figured out as we talk.
- This session is a demo of lots of basics.
  - Ask questions - otherwise it will be boring
  - If you want to use this in real life: you will need *mentoring sessions* and active help. Contact us to do that.

Warning

Audio and video weirdness

We are setting up video and audio recording, but also capturing that for re-sharing it. There may be anomalies as video or audio feedback. Please be prepared.

Why we stream

Objectives

- Learn the general history of CodeRefinery streaming.
- Discuss the benefits of streaming and recording
- Discuss the downsides and difficulties

Instructor note

- Discussion: 10 min
- Q&A: 5 min
- Exercises: 0 min

This is a general discussion of the topics of the day, focused on the history of why we stream and record, how we got here, and how people feel about it. We won't focus on how anything is done.

Icebreaker questions

- What is the most people you have taught for?
- What are the distinct parts of teaching, that can be separated? (teaching, helping, etc)

What is streaming and recording?

- Streaming is mass communication: one to many
  - Interaction audience→presenters is more limited (but different)
- Using consumer-grade tools, normal people can reach huge audiences by Twitch/YouTube/etc.
- This isn't actually that hard: people with much less training than us do it all the time.
- They reach huge audiences and maintain engagement for long events.

Recording and rapid video editing is useful even without streaming.

History

- In-person workshops
  - 3 × full day, required travel, infrequent, one-shot
- Covid and remote teaching
  - Traditional "Zoom" teaching several times
- Mega-CodeRefinery workshop
  - 100-person Zoom teaching
  - Emphasis on teams
- Research Software Hour
  - Livestream free-form discussions on Twitch
- Streamed "HPC Kickstart" courses

Benefits and disadvantages

Benefits:

- Larger size gives more (but different) interaction possibility
  - "Notes" for async Q&A
- Recording (with no privacy risk) allows instant reviews
- Stream-scale allows for many of the things you have learned about in days 1-3.

Disadvantages:

- Requires training for using the tools
- Requires a certain scale to be worth it
- Coordination is much harder for big events

When would I recommend it?

- No: For small courses
- Questionable: Medium sized courses with no videos
- Yes: When you want the largest audience
- Yes: When you want without registration required

- Yes: When you want good reusability/fast videos

Future prospects (briefly)

- Streaming probably stays as a CodeRefinery tool
- We *can* scale our courses much larger than they are now. Why don't we, together with others?
- These tools are useful in other places too
  - I've used them to record my own talks to make videos of my single-person presentations or record conference talks nicer than Zoom.

Q&A

Q&A from audience

Keypoints

- Streaming optimizes a course for different things
- The disadvantages can be compensated for
- There are benefits and disadvantages

Behind the stream

Objectives

- Take a first look at the broadcaster's view.
- Get to know what happens "behind the stream" of a workshop
- See what the "broadcaster" sees and what they need to do.
- Not yet: learn details of how to do this.

Instructor note

- Teaching: 20 min
- Q&A 10 min

In this episode, you'll see an end-to-end view of streaming from the broadcaster's point of view. It's a tour but not an explanation or tutorial.

Who does what

We have certain role definitions:

- **Broadcaster**: Our term for the person who manages the streaming.
- **Director**: Person who is guiding the instructors to their sessions, changing the scenes, calling the breaks, etc.
  - Could be the same as broadcaster.

- **Instructor**: One who is teaching. They don't have to know anything else about how streaming works.

This lesson describes what the Broadcaster/Director sees.

Window layouts

What does the broadcaster see on their screen?

- What are the main windows you see?
- What do each of them do?
- Which ones do you need to focus on?
- How do you keep all this straight in your head?

CodeRefinery control panel

- A custom application that controls scenes
- Based on OBS-websocket (remote control connection for OBS - we'll learn about this later)
- Can also work remotely, so that you can have a remote director

How scenes are controlled

What has to be done during a course?

- How do you start the stream?
- How do you change the view?
- How do you adjust things based on what the instructors share?
- How do you coordinate with the instructors?
- How do you know when to change the view?

Getting it set up

- How hard was it to figure this out?
- How hard is it to set it up for each new workshop?

What can go wrong

- What's the worst that has happened?
- What if you need to walk away for a bit?
- Someone broadcasts something unexpectedly

Alternatives

- Youtube vs Twitch

- Zoom stream directly to YouTube/Twitch
- Direct streaming platform, e.g. streamyard

Q&A

Q&A from audience

Keypoints

- The broadcaster's view shouldn't be so scary.
- There is a lot to manage, but each individual part isn't that hard.

Video editing

Objectives

- Get to know ways of quick video editing to be able to provide accessible videos
- Learn how video editing can be distributed and done the same day.

Instructor note

- Teaching: 20 min
- Exercises: 20 min

Video recordings could be useful for people watching later, but also are (perhaps more) useful for **immediate review and catching up after missing a day in a workshop**. For this, they need to be released immediately, within a few hours of the workshop. CodeRefinery does this, and you can too.

Hypothesis: videos must be processed the same evening as they were recorded, otherwise (it may never happen) or (it's too late to be useful). To do that, we have to make processing *good enough* (not perfect) and *fast* and the work *distributeable*.

Primary articles

- Video editor role description: https://coderefinery.github.io/manuals/video-editor/
- ffmpeg-editlist: the primary tool: https://github.com/coderefinery/ffmpeg-editlist
  - Example YAML editlists: https://github.com/AaltoSciComp/video-editlists-asc

How this relates to streaming

- If you stream, then the audience *can not* appear in the recorded videos
- This allows you to release videos *very quickly* if you have the right tools.
- When you have a large audience, the videos start helping more (review a missed day, catch up later, review later)
- Thus

    - If you would never want videos, there may never be a benefit to streaming
    - If you want videos, it gives motivation to stream.

Summary

- Basic principle: privacy is more important than any other factor. If we can't guarantee privacy, we can't release videos at all.
    - Disclaimers such as "if you don't want to appear in a recording, leave your video off and don't say anything", since a) accidents happen especially when coming back from breakout rooms. b) it creates an incentive to not interact or participate in the course.
- Livestreaming is important here: by separating the instruction from the audience audio/video, there is no privacy risk in the raw recording. They could be released or shared unprocessed.
- Our overall priorities
    1. No learner (or anyone not staff) video, audio, names, etc. are present in the recordings.
    2. Good descriptions.
    3. Removing breaks and other dead time.
    4. Splitting videos into useful chunks (e.g. per-episode), perhaps equal with the next one:
    5. Good Table of Contents information so learners can jump to the right spots (this also helps with "good description".)
- ffmpeg-editlist allows us to define an edit in a text file (crowdsourceable on Github), and then generate videos very quickly.

How we do it

The full explanation is in the form of the exercises below. As a summary:

- Record raw video (if from a stream, audience can't possibly be in it)
- Run Whisper to get good-enough subtitles. Distribute to someone for checking and improving.
- Define the editing steps (which segments become which videos and their descriptions) in a YAML file.
- Run ffmpeg-editlist, which takes combines the previous three steps into final videos.

Exercises

Exercise A

These exercises will take you through the whole sequence.

✍ Editing-1: Get your sample video

Download a sample video:

- Video (raw): http://users.aalto.fi/~darstr1/sample-video/ffmpeg-editlist-demo-kickstart-2023.mkv

- Whisper subtitles (of raw video): http://users.aalto.fi/~darstr1/sample-video/ffmpeg-editlist-demo-kickstart-2023.srt
- Schedule of workshop (day 1, 11:35–12:25) - used for making the descriptions. :::::

✍ Editing-2: Run Whisper to generate raw subtitles and test video.

First off, install Whisper and generate the base subtitles, based on the. Since this is probably too much to expect for a short lesson, they are provided for you (above), but if you want you can try using Whisper, or generating the subtitles some other way.

You can start generating subtitles now, while you do the next steps, so that they are ready by the time you are ready to apply the editlist. ffmpeg-editlist can also slice up the subtitles from the main video to make subtitles for each segment you cut out.

Whisper is left as an exercise to the reader.

✔ Solution

Example Whisper command:

```
$ whisper --device cuda --output_format srt --initial_prompt="Welcome to
CodeRefinery day four." --lang en --condition_on_previous_text False
INPUT.mkv
```

An initial prompt like this make Whisper more likely to output full sentences, instead of a stream of words with no punctuation.

✍ Editing-3: Create the basic editlist.yaml file

Install ffmpeg-editlist and try to follow its instructions, to create an edit with these features:

- The input definition.
- Two output sections: the “Intro to the course”, and “From data storage to your science” talks (Remember it said the recording started at 11:35… look at the schedule for hints on when it might start!). This should have a start/end timestamp from the *original* video.

A basic example:

```
- input: day1-raw.mkv

# This is the output from one section.  Your result should have two of these
sections.
- output: part1.mkv
  title: something
  description: >-
    some long
    description of the
    segment
  editlist:
    - start: 10:00    # start timestamp of the section, in *original* video
    - end: 20:00      # end timestamp of the section, in the *original* video
```

✔ Solution

This is an excerpt from our actual editlist file of this course

```
- input: day1-obs.mkv

- output: day1-intro.mkv
  title: 1.2 Introduction
  description: >-
    General introduction to the workshop.

    https://scicomp.aalto.fi/training/kickstart/intro/

  editlist:
  - start: 00:24:10
  - end: 00:37:31


- output: day1-from-data-storage-to-your-science.mkv
  title: "1.3 From data storage to your science"
  description: >-
    Data is how most computational work starts, whether it is
    externally collected, simulation code, or generated. And these
    days, you can work on data even remotely, and these workflows
    aren't obvious. We discuss how data storage choices lead to
    computational workflows.

    https://hackmd.io/@AaltoSciComp/SciCompIntro

  editlist:
  - start: 00:37:43
  - end: 00:50:05
```

💬 Discussion: what makes a video easy to edit?

- Clear speaking and have high audio quality.
- For subtitle generation: Separate sentences cleanly, otherwise it gets in a “stream of words” instead of “punctuated sentences” mode.
- Clearly screen-sharing the place you are at, including section name.

- Clear transitions, “OK, now let’s move on to the next lesson, LESSON-NAME. Going back to the main page, we see it here.”
- Clearly indicate where the transitions are
- Hover mouse cursor over the area you are currently talking about.
- Scroll screen when you move on to a new topic.
- Accurate course webpage and sticking to the schedule

All of these are also good for learners. By editing videos, you become an advocate for good teaching overall.

✍ Editing-4: Run ffmpeg-editlist

Install ffmpeg-editlist: `pip install ffmpeg-editlist[srt]` (you may want to use a virtual environment, but these are very minimal dependencies).

The `ffmpeg` command line tool must be available in your `PATH`.

✔ Solution

It can be run with (where `.` is the directory containing the input files):

```
$ ffmpeg-editlist editlist.yaml .
```

Just running like this is quick and works, but the stream may be garbled in the first few seconds (because it’s missing a key frame). (A future exercise will go over fixing this. Basically, add the `--reencode` option, which re-encodes the video (this is **slow**). Don’t do it yet.

Look at the `.info.txt` files that come out.

✍ Editing-5: Add more features

- Several chapter definitions.(re-run and you should see a `.info.txt` file also generated). Video chapter definitions are timestamps of the *original* video, that get translated to timestamps of the *output* video.

```
- output: part1.mkv
  editlist:
  - start: 10:00
  - -: Introduction    #  <-- New, `-` means "at start time"
  - 10:45: Part 1      #  <-- New
  - 15:00: Part 2      #  <-- New
  - end: 20:00
```

Look at the `.info.txt` files that come out now. What is new in it?

- Add in “workshop title”, “workshop description”, and see the `.info.txt` files that come out now. This is ready for copy-pasting into a YouTube description (first line is the title, rest is the description).

  Look at the new `.info.txt` files. What is new?

✔ Solution

- This course actually didn’t have chapters for the first day sessions, but you can see chapters for day 2 here, for example.
- Example of the workshop description for this course
- Example info.txt file for the general introduction to the course. The part after the `-----` is the workshop description.

```
1.2 Introduction - HPC/SciComp Kickstart summer 2023

General introduction to the workshop.

https://scicomp.aalto.fi/training/kickstart/intro/

00:00 Begin introduction       <-- Invented for the exercise demo, not real
03:25 Ways to attend           <-- Invented for the exercise demo, not real
07:12 What if you get lost     <-- Invented for the exercise demo, not real

-----

This is part of the Aalto Scientific Computing "Getting
started with Scientific Computing and HPC Kickstart" 2023
workshop.  The videos are available to everyone, but may be
most useful to the people who attended the workshop and want
to review later.

Playlist:
https://www.youtube.com/playlist?list=PLZLVmS9rf3nMKR2jMglaN4su3ojWtWMVw

Workshop webpage:
https://scicomp.aalto.fi/training/scip/kickstart-2023/

Aalto Scientific Computing: https://scicomp.aalto.fi/
```

✍ Editing-6: Subtitles

Re-run ffmpeg-editlist with the `--srt` option (you have to install it with `pip install ffmpeg-editlist[srt]` to pull in the necessary dependency). Notice how `.srt` files come out now.

Use some subtitle editor to edit the *original* subtitle file, to fix up any transcription mistakes you may find. You could edit directly, use `subtitle-editor` on Linux, or find some other tool.

What do you learn from editing the subtitles?

✔ Solution

```
$ ffmpeg-editlist --srt editlist.yaml
```

There should now be a `.srt` file also generated. It generated by finding the `.srt` of the original video, and cutting it the same way it cuts the video. Look and you see it aligns with the original.

This means that someone could have been working on fixing the Whisper subtitles while someone else was doing the yaml-editing.

✍ Editing-6: Subtitles

Re-run ffmpeg-editlist with the `--srt` option (you have to install it with `pip install ffmpeg-editlist[srt]` to pull in the necessary dependency). Notice how `.srt` files come out now.

Use some subtitle editor to edit the *original* subtitle file, to fix up any transcription mistakes you may find. You could edit directly, use `subtitle-editor` on Linux, or find some other tool.

What do you learn from editing the subtitles?

✍ Editing-7: Generate the final output file.

- Run ffmpeg-editlist with the `--reencode` option: this re-encodes the video and makes sure that there is no black point at the start.
- If you re-run with `--check`, it won't output a new video file, but it *will* re-output the `.info.txt` and `.srt` files. This is useful when you adjust descriptions or chapters.

💬 Discussion: how to distribute this?

Create a flowchat of all the parts that need to be done, and which parts can be done in parallel. Don't forget things that you might need to do before the workshop starts.

How hard was this editing? Was it worth it?

Exercise B

This is a more limited (and older) version of the above exercise, using an synthetic example video.

✍ Use ffmpeg-editlist to edit this sample video

Prerequisites: `ffmpeg` must be installed on your computer outside of Python. Be able to install ffmpeg-editlist. This is simple in a Python virtual environment, but if not the only dependency is `PyYAML`.

- Download the sample video: http://users.aalto.fi/~darstr1/sample-video/sample-video-to-edit.raw.mkv
- Copy a sample editlist YAML
- Modify it to cut out the dead time at the beginning and the end.
- If desired, add a description and table-of-contents to the video.
- Run ffmpeg-editlist to produce a processed video.

✔ Solution

```
- input: sample-video-to-edit.raw.mkv
- output: sample-video-to-edit.processed.mkv
  description: >
  editlist:
    - start: 00:16
    - 00:15: demonstration
    - 00:20: discussion
    - stop: 00:25
```

```
$ ffmpeg-editlist editlist.yaml video/ -o video/
```

Along with the processed video, we get `sample-video-to-edit.processed.mkv.info.txt`::

```
This is a sample video


00:00 Demonstration
00:04 Discussion
```

See also

- ffmpeg-editlist demo: https://www.youtube.com/watch?v=thvMNTBJg2Y
- Full demo of producing videos (everything in these exercises): https://www.youtube.com/watch?v=_CoBNe-n2Ak
- Example YAML editlists: https://github.com/AaltoSciComp/video-editlists-asc

Keypoints

- Video editing is very useful for learning
- Set your time budget and make it good enough in that time
- Reviewing videos improves your teaching, too.

Open Broadcaster Software (OBS) introduction

Objectives

- Understand that OBS is a video mixer.
- Understand the basic controls and features of OBS.

Instructor note

- Teaching: 15 min
- Q&A 5 min

- OBS theory in CodeRefinery manuals
- The next episode Open Broadcaster Software (OBS) setup

In this episode, you'll get more familiar with the OBS portion of the streaming setup. You'll see how it's used, but not yet how to configure it from scratch. You'll learn how to be a "director".

What is OBS?

- Formally "OBS Studio"
- Most commonly known as a livestreaming application.
- Open source, free.
- Cross-platform, easy to use screencasting and streaming application.
- Real-time video mixer.

OBS user interface

- We'll click through each view.
- What does each view do?
- Let's click through the buttons.

- Let's see the important config options.

OBS during a course

- What management is needed.
- The control panel.
- Audio.
- Adjusting windows and so on.

Hardware requirements

- Reasonably powerful broadcast computer
  - CodeRefinerys streaming computer is 8 CPU AMD, 64GB memory
  - That is way overkill: a powerful laptop can probably do this.
- Large second monitor for laying out the windows you capture
- Stable internet connection (wired preferable)
  - CodeRefinery's broadcast is the slowest purchasable: 100 Mbit down / 25Mbit up

Q&A

We'll answer audience questions.

See also

- The next episode Open Broadcaster Software (OBS) setup which is about configuring OBS.

Keypoints

- OBS may seem complicated, but it's a graphical application and most pieces make sense once you know how it works.

Open Broadcaster Software (OBS) setup

Objectives

- See how to configure OBS using the pre-made CodeRefinery scene collections
- Modify the collections to suit your needs.

Instructor note

- Teaching: ?? min
- Hands-on: ?? min
- Q&A: ?? min

- The previous episode Open Broadcaster Software (OBS) introduction

- OBS theory in CodeRefinery manuals

In this lesson, we'll see how to configure OBS from scratch for your purposes. We'll do this by deleting the instructor's configuration and trying to recreate it

This section is short, since it has never been done before: we'll just give it a short and update the lesson later.

CodeRefinery OBS configs

- CodeRefinery configs are shared in a repository: https://github.com/coderefinery/obs-config
- These can be imported to pre-configure some things
- There are two types of configs:
  - Profiles
    - Servers, resolutions, audio, video, etc.
  - Scene collections
    - The graphical layouts.

Installing the OBS config

- Clone the git repository.
- Import the profile and scene collections under their respective menus.

Initial setup

- Click through each menu and change anything that is needed
- Set the streaming server

👀 The instructor will go through the setup.

- Reset configuration: `mv .config/obs-studio/ .config/obs-studio-old/`
- Import `profiles/TeachingStreamingv3/`
- Import `scenes/TeachingStreamingZoomv3.json`

Set up the remote control

- Create a Python environment and install it: `pip install https://github.com/coderefinery/obs-cr/archive/refs/heads/master.zip`
- Run it: `python obs_cr/control.py localhost:4445 TOKEN --broadcaster`
- There are more options but let's not cover them yet... and leave this for a hands-on session.

Setup before each course

- Re-confirm audio
- Re-confirm each scene
- Test everything
- rkdarst has a rather long checklist, but each individual step is short.

Q&A

We'll answer audience questions.

Keypoints

- Most of our configuration has been OBS may seem complicated, but it's a graphical application and most pieces make sense once you know how it works.

What's next?

Objectives

- Know next steps if you want to do streaming

Instructor note

- Teaching: 5 min
- Q&A 5 min

What comes next?

- We talked a lot about theory, and gave demonstrations.
- Hands-on is very different. We recommend working with someone to put it in practice.
-

- Work with someone who can show you the way
- Use it for smaller courses with a backup plan

See also

Keypoints

- These lessons about streaming have been the theoretical part of streaming training.

Livestream practice

This page contains various exercises to be done during the practice session. These are all "roleplay" of how you'll actually act in a livestreamed course.

Audio exercises.

Do the bottom two exercises from Sound. Audio quality and balance is an important prerequisite for any teaching.

Screenshare check

Purpose: be able to share screen in a way that can be broadcasted. Remember this size so that it can be set up quickly later - you'll need to do this quickly, without hesitation, later on.

Background: most Zoom instances allow you to "share a portion of the screen", which should be 840 wide × 1080 tall. This is rendered pixel-perfect by YouTube and also forces a "small screen" so that you don't share too much, and learners have space on their own screens.

✍ Demonstrate a proper screenshare

- Share your screen and adjust for portrait mode.
- If you can't share a portion of the screen, share your whole screen and tell the director what your total resolution is.
- Director: program in this as a named scene
- Instructor: Open the preview pane to see how it looks
- Instructor: make a note (tape on your screen?) to indicate the amount of space you have.

Behind the livestream

We'll see demos of what the streaming part is like.

Director view

This is currently done case-by-case/live demo during the exercises below.

Announciator panel

We have a web-based panel that can relay non-verbal messages ("caution", "warning", "faster", etc.) between instructors without having to speak up. This exists as a web panel that has various lights light up and make sounds when clicked.

✍ Test the announciator panel

Purpose: Understand how non-verbal instructor feedback works.

- Open the announciator panel (the broadcaster will provide a link)
- Make it work (disable SSL, click the button to enable audio). Confirm that it's lighting up.
- understand:
  - The indicator buttons and what they mean.
  - What effects these buttons have and when to use them.
- All: test turning the various lights on and off.

Roleplay:

- Person A explains something.
- Person B, partway through it, lights up the "time" light.
- Person A notices

Going live

Getting the instructors ready to "go live" isn't hard but someone has to "herd the cats" and make it happen. Instructors need to know how this process goes.

✍ Test the "going live" process

- Director: Announce it's almost time
- Director: asks everyone else in the call to turn off their videos.
- Decide who is sharing screen
- Decide who is speaking first when you go live and what the first words will be. (These first words are most important and you should think of what you'll say.)
- Director: In the director's panel, configure the "back to" setting, broadcaster audio, and jingle
- Director: countdowns "Starting in 3 2 1", (click the button), (everyone takes a breath for zero), then go.
- Speaker begins talking
- (Remember that each time you go back live is a good time to review the notes + answer questions)

Introduce exercises

It's not that hard to teach, but it requires some practice to say the first words of a new lesson smoothly.

✍ Exercise

Purpose: Move to exercises in a controlled manner.

- Say what exercises are
  - What your goal is
  - How far you should get
  - Common things that might go wrong
- Add the exercise info + above questions to the notes
- Answer any questions from the notes (this)
- Verify that it is in the notes
- Leave to the break.

Going to a break

✍ Go to a break

- Look at the notes, anything to be answered? (At least let people know it's in use)
- Say when we come back and what to expect then.
- Say "bye".
- Director pushes the "BEAK" button.
- Update the notes:
  - Break info block
  - When back
  - Exercise link (if any)
  - What the expectations are

Notes fixup

Purpose: learn the common things that can go wrong in the notes and fix them.

✍ Practice fixing up a notes document

- spacing is off (missing blank lines between questions, missing blank lines in bullet points)
- questions not answered
- Section headings at inconsistent levels
- Exercises not described once they start
- Lesson/episode titles are not there
- People asking/answering questions above the exercise definition
- No template lines at the bottom.

Roleplay: one person teaches, several people write badly in Notes, trainees have to fix up in real time

✍ Add common notes blocks

In the notes, everyone add examples of the below, and we'll compare the different formats people create (and maybe get some new ideas):

- Exercise announcement
- Break announcement
- Poll of some sort

Evaluate not just on how good it looks, but also how long it takes to make, risks of things going wrong, brevity of markdown+HTML versions, etc.

Lesson preparation

✍️ Level zero prepration to teach a lesson

- Explain the point of a lesson in a few sentences (as if you are giving an introduction to learners)
- Explain what you will not cover

Co-instructing examples

- Interrupt to bring up questions
- Ask for a notes check

Lesson revision

- Revise this lesson to make it more usable for someone who doesn't know the command line (or is in some other way different from you).

Streaming practice 1

This series of exercises introduces you to the practices of livestream teaching. It's nothing that radical, but since you don't directly see the audience, you need to interact with your co-instructor to make the session appear engaging.

Basics of this lesson:

- There are four role-playing scenarios.
- First, there is a demo and discussion together.
- Then, go to breakout rooms to practice with a partner. Practice with each person in each role.
- Then, we come back to the main room and see polished demos (hopefully from each group, but we'll see the time)

Not in scope for this lesson:

- Actual lesson content (use the Sample teaching plan or a lesson you already know or make up something that sounds good for the purposes of this lesson).
- How to teach well.
- Technical aspects of making a good screenshare and audio quality.

- Director and broadcaster related things (but someone will simulate that for you).

Going live

Purpose: saying the first words after a session starts.

You may think that starting a session isn't hard, and it isn't. But if it's extra smooth, it leaves a good first impression. Instructors also often make the mistake of going straight to the topic and not framing the whole session at the start.

You want to prepare so that once you go live, it's all smooth. The things you need to consider:

- During the break, review notes and what will happen in the upcoming session.
- Get everyone not active to turn off their videos.
- Decide who speaks first.
- Decide who does what over the next session. (talker, typer, primary, secondary, etc.)
- **Give a natural, motivating intro of the whole lesson, suitable for a wide audience. It should set the expectations for what comes next.** Start with a minute or two of broad intro before getting into the actual technical topic.

The roles:

- Director: person pushing the buttons to go live (could be one of the speakers). Counts down 3 [pushes "back"], 2, 1, [pause].
- Instructor A: speaks first and leads in to the main program. Engages instructor B.
- Instructor B: engages when it's time.

❗ The "going live" process with the director

- (Director: Announce it's almost time)
- (Director: asks everyone else in the call to turn off their videos.)
- Decide who is sharing screen
- Decide who is speaking first when you go live and what the first words will be. (These first words are most important and you should think of what you'll say.)
- Director: In the director's panel, configure the "back to" setting, broadcaster audio, and jingle
- Director: countdowns "Starting in 3 2 1", (click the button), (everyone takes a breath for zero), then go.
- Speaker begins talking
- (Remember that each time you go back live is a good time to review the notes + answer questions)

✍️ Exercise Live-1: Begin a session smoothly (10 min)

You will practice going live (but without the director part) in breakout rooms. In the main room, a few groups will demo and a director will cue you to start. Swap and try again

- Break into groups of two
- Choose a lesson to practice. For example (not limited to this):
  - Sample teaching plan
  - https://coderefinery.github.io/git-intro/browsing/
  - https://coderefinery.github.io/testing/
  - You could do something else, too.
- Practice giving the first two minutes of the lesson. The emphasis is doing it several times to make it feel smooth.
- A good strategy:
  - Plan it all in advance
  - Decide who speaks first (A)
  - Simulate a countdown "3… 2… 1… [pause]… [begin]"
  - A starts speaking and asks a question of B to engage them.
  - Brief intro discussion before moving to the technical topics.
  - Stop once you get to the technical topic. ("OK, let's move on to the main material now.")

Doing a demo

Demos work great with co-teaching. In this session, we will practice the "talker/typer" system: the **talker** explains what is going on while the **typer** does the typing.

Team teaching via talker and typer.

"Talker" and "typer" is one way of looking at roles for a session. This is separate from "**primary**" (keeps time and ensures overall flow) and "**secondary**" in lesson preparation. For example, the primary can be the typer, give the intro, and then ask the typer "so, explain to me how we do X."

See more: Co-teaching

Talker responsibilities:

- Manage the overall flow and timing. They can keep an eye on the lesson and clock on their screen.
- Carefully say what the typer should do. Describe it clearly, as if the typer was a student and didn't know the topic already.
- Ask the typer "what do you think?" to keep a conversation going
- Ask the typer "what should come next?" if they ever get lost or wonder what comes next.

Typer responsibilities:

- Does what the talker says (this slows things down and forces everything to be said). Don't rush ahead!
- Should ask questions about confusing points or if the talker makes misses some steps that the learners need to hear. This keeps a conversation going.
- Jump in to remind the talker about what should go next, etc.

✍ Exercise Demo-1: Practice a demo (10 min)

Stay in your groups of 2. Practice doing a demo for 2-3 minutes. Swap and try again. This isn't a lot of time, but try to at least:

- Divide to talker and typer.
- Do the demo in takler/typer format as described above.
  - You can use the same sample lessons mentioned above
- Typer should ask at least one clarification from the talker.
- Talker should ask at least one question to the typer.

Q&A

Q&A is via the notes document, which can be just as or even more powerful than a live Q&A. In this exercise, we do a practice Q&A session.

Considerations:

- Screenshare the notes so that the audience can see what you are discussing. (It's good to do this periodically even if there aren't questions, so people will know it's being monitored.)
- Try to involve both co-instructors. For example, one person can find an interesting question and first ask the other person what they think before giving their own answer.
- If there are many questions you may divide up one person mainly looking for question (and talking sometimes) and the other mainly answering (and looking sometimes)

✍ Exercise Notes-1: Simulate a Q&A via Notes

Divide into groups of **three** for this one.

The goal is to have two people discussing questions in a notes document while another is adding questions live. (The third person can also possibly be answering some of the question to simulate other helpers). The Sample teaching plan has some sample questions.

Expected steps:

- One instructor shares the notes
- Instructors scroll through and discuss the questions
  - Keep in mind that the answers to some questions can be "we will discuss later".

- (If using the indicator panel) End once “time” is called via the control panel indicator light.
- If there is a third person and using live notes, they can go answering questions at the same time the instructors are talking.

Sample questions you can be asked:

- If you have a “sample audience”, they can ask questions in a notes dock from the Sample teaching plan.
- If you are only two people, you could use notes from one of old CodeRefinery workshops (pretend you are in the workshop, helpers have already answered many questions, and you are following up on important things in the stream):
  - Day 1, committing: https://coderefinery.github.io/2025-03-25-workshop/questions/day1/#commiting
  - Day 2, inspecting history: https://coderefinery.github.io/2025-03-25-workshop/questions/day2/#inspecting-history

Saying bye

Going to a break/exercise isn’t hard. But, since there’s no opportunity to ask more questions, you should make sure that everything is done.

Considerations:

- Give a wrap-up of the session that is just ending, so that people can unpack.
- Give clear directions of what learners should do next, for example:
  - Do an exercise (state the goals which should be accomplished by the end), or
  - Take a break and don’t forget to walk around some.
  - When to be back.
  - Update the notes with the info.
- Someone will be waiting to switch the livestream to break mode. They can’t read your mind, so **someone should say “bye”** to indicate the session is over. Then, the director will immediately switch to break mode.

✍ Exercise Bye-1: Wrap up and say bye smoothly

Simulate a session ending according to the things above. It’s your choice what session, but here are some suggestions:

- Sample teaching plan
- https://coderefinery.github.io/git-intro/commits/
- https://coderefinery.github.io/testing/test-design/
- Anything you have done

Try to include the following bits:

- Summarize what was just (pretended to be) learned
- Say what comes next (break or exercise)
- Some small interaction between co-instructors (asking each other a question)
- Whoever is talking least adds a description in the notes of what comes next.

Example notes description for going to an exercise:

```
:::info
### Exercise Cloning until xx:12.
https://coderefinery.github.io/git-intro/local-workflow/#exercise

* Try to complete at least steps 1-7.  We will be using this in the next exercise.

:::
```

Summary

We went over some of the most common things you'll experience when co-teaching via livestream. Think about these when preparing your lessons to teach.

Preparation

Motivational video

Watch this 10 minute motivation video that goes over some of the major things we need to think about when preparing to teach.

You can also read the transcript below if you want (but note that CATS, CodeRefinery's unofficial mascot, makes an appearance in the video).

https://youtu.be/61Cdi4Eje2Y

Transcript

Hi, I was about to play a board game with some friends here, and it reminded me of how similar I see the process of teaching board games to teaching computer technology, like we do in CodeRefinery. So I thought I would make a quick introduction to our instructor training by talking about board games.

So when teaching a board game, **the first thing is I want to make sure I know the game well enough to teach it. And that's different than knowing the game very well. When teaching, you need to know it well enough to know what is not important to teach and what you need to focus on. So the game that a new player is playing is very different than the game that an experienced player is teaching.** And it requires a lot of skill and preparation to know this.

Next up, **I'll make sure the setup is good.** So I'll try to set it up before I expect everyone to be sitting down and watching everything I'm doing, because that's just taking their time and attention.

Next up when teaching, **the first thing I want to make sure I get clear is the theme of the game.** So that means within the game, who are we as characters in the game? **What are we trying to do and why?** It's very easy to get started and talk about all the little mechanics of how things move and how to accumulate points and so on without understanding what the purpose is. Like, where's the fun in here what's um what are we trying to accomplish in the game? Without knowing this kind of thing it's hard to really have engagement and for the computer technology teaching, the same goes there. It's very often that people teach the mechanics of what to do without giving the big picture of why, and this is quite a big problem.

So next up I'll try to do as much teaching as possible while we're actually doing things. We'll get to this a little bit later, but the teaching will continue even after we've started the game. I'll try to get the game started as soon as possible. People can investigate things, fiddle around with it while the teaching's going on. I'll try to be moving it while I'm teaching, and so on. So for the technology thing, this clearly has good examples that people can work on at the same time. When teaching the computer stuff, it's important to give them time to do their own things and not overload them with what you're saying.

Next up, and this is a tricky one, **there should be one teacher. So as you may know from our CodeRefinery teaching, we always have multiple teachers. But there should be one person whose responsibility it is to make sure that the time is being kept and the lessons progressing** and people are learning what they need. And then co-instructors support them. So for example when teaching the game I might be explaining what's going on and I'll get someone else that knows to be moving the things and pointing to stuff as I am doing the teaching. So in CodeRefinery this doesn't mean there's the main teacher and the second teacher. Oftentimes they're equal, but they agree to hand off different parts of the lesson. So that way there's a different primary teacher at different times. This is important to discuss in advance. When there's multiple teachers or when there's one teacher and other people that know the game, it's really important that there's not other people that are interrupting the flow, bringing and saying, oh, don't forget this, don't forget, don't forget that, and so on. Because maybe that's planned to come later on where it's more natural or maybe I've decided that this shouldn't be taught at all for the first game, because of just making things more complex. So this is something that needs to be discussed with the co-teachers and make sure that you have the overall plan set up. Like I said, it's not the primary teacher and secondary but it's two people working together to make a course, and the flow control is by one person at a time.

Okay next stop. **Make it easier.** As I said at the beginning you should know the game enough to know how to make it easier so the things that don't really matter for the first game aren't [taught]. I might just not even teach or I might begin the game a few rounds ago and then I start introducing these things or introduce them in a second game or so on.

So this is to make it the game that the learners will play and not the game that I would play. There's other ways to do this. So for example, the first game may essentially be cooperative where we're all helping each other. We're playing with open hands and so on. And then we're all giving advice for each person when it's their turn. When I'm making my moves, I'll take time to explain what my long-term strategy is, even if I wouldn't do that during a normal game. Speaking of strategy, I'll often try to explain the broad strategy at the beginning. So basically, for example, try to create paths across the whole game board at the beginning, and so on. I'll try to not get really involved in the micro strategy of how to optimize each individual turn and so on because that just takes a lot of time. So as it goes, broad strategy and then focus on the small scale movements and technical details and things like that, but not small-scale strategy.

Okay, next up, when teaching, **I'll always make sure experienced people go first. I never expect someone to make a move or do something until they've seen someone else do it first.** So I'll arrange the game so that the experienced players sit first in turn order, and then come the learners so that way by the time it's any learner's turn they've seen a few complete rounds and have heard the people explain what they're doing and why. That lets us get started as soon as possible.

Next up I'll be very careful about jargon. The players know these are markers and these are wooden sticks and so on, but they don't know intuitively in the game that these are tracks and these are stations. So I'll try to make sure that I explain something that makes sense to them and then throw in the jargon at the same time so that way they can connect the real world piece to the in-game concept.

Okay, then when teaching, and this doesn't really apply to board games that much, **but I'll try to at least do one run through [of the teaching] myself.** So I don't need to do extensive preparations, but I want to make sure that I understand the general flow of what's going on and I've at at least typed it out [the examples] some before and given demo. **So I'll especially focus on the first five minutes and the last five minutes of the lesson,** because the first five minutes is this theme, who we are, what we're doing, and why it matters. And that's often missing from the technology teaching. So [the learner] learns about what it is but [they] don't really understand why [they are] learning it, and that's a big problem. And then at the end, I'll […] really try to rehearse the explaining of, okay, what we just learned and what comes next, and what you would do in your future games and so on, and how it gets more interesting. And if something is not perfect in the middle of the teaching, well, that's just a learning experience so you can explain: "oh yeah I did something wrong, and here's why it's wrong, and here's how I recovered from it". Perhaps more relevant for the technology teaching but possibly for board games also.

And then finally the teaching isn't just when the lesson is ongoing. There'll be the lesson at the beginning and we start playing the game **but the mentoring is part of the teaching and that goes on throughout the entire first game and possibly future games.** This is why in CodeRefinery we have such an emphasis on things like local breakout rooms or local

helpers, if we can find them. Because any kind of course can just be an introduction and it's the working with people together over a longer time that really helps people to learn well and apply it in practice.

So with that being said, I hope that this has given you some insights and we'll see you in a CodeRefinery instructor training sometime and we will get to our game now. So thanks a lot. Bye.

(try to watch/read before reading more)

Teaching online in CodeRefinery is something really unique. The basic lesson and teaching style is a lot like you might expect from Carpentries or other practical teaching, but our online teaching really is different. This video doesn't go into online teaching, but it'll come.

Some extra points not emphasized enough in the video:

- Adopting the learning perspective can sometimes be the hardest part. We spent a lot of time designing our lessons to do this, and it takes many revisions (and times change and it needs to be updated).
- We'd like to re-emphasize the "tell the broad theme" part. People are smart and can read more later, if they know why they should.
- Co-teaching can't be emphasized enough. We always do this, and teachers work together to plan, even if one is still learning. Still, it should be clear who is doing the flow control at any given time.
- The "always show a demo first" is different from academic courses. Most of our audience want to figure out a tool to use right away, not get more mental work.
- Mistakes are OK! You have a co-instructor to help you figure out and you get to teach debugging and not being afraid of errors.

Collaborative notes archives from workshops

We have collected archives of the notes from past workshops. These are only available in the primary GitHub Pages build of the workshop site (and not, for example, the PDF or single-page HTML builds).

- 2024

Instructor guide

Target audience

- Everyone teaching online workshops about computational topics or interested in teaching.
- Previous and future instructors and helpers of CodeRefinery workshops.

Timing

Session 1

- 15 min: Intro
- 45 min: Lesson design and development
- 15 min: Break
- 45 min: Lesson template
- 15 min: Break
- 30 min: How we collect feedback and measure impact
- 10 min: Outro and feedback

Session 2

- 15 min: Intro
- 10 min: About the CodeRefinery project and CodeRefinery workshops in general
- 30 min: Collaborative notes and interaction
- 15 min: Break
- 35 min: Workshop overview, roles, onboarding/installation, helpers
- 20 min: Sound
- 15 min: Break
- 30 min: Screenshare
- 10 min: Outro and feedback

Session 3

- 15 min: Intro
- 30 min: Computational thinking
- 15 min: Break
- 30 min: Teaching philosohies
- 30 min: Co-teaching
- 15 min: Break
- 35 min: Sharing teaching tips, tricks, tools etc
- 10 min: Outro and feedback

Session 4

- 15 min: Intro
- 10 min: Why we stream
- 20 min: Behind the stream
- 15 min: Video editing (part 1)
- 15 min: Break
- 25 min: Video editing (part 2, exercise)
- 20 min: OBS introduction
- 15 min: Break

- 30 min: OBS setup
- 15 min: What's next?

Talking points for each sessions intro

- Brief CodeRefinery intro
- Instructors intros
- Notes intro
  - Check-in
  - Icebreaker
  - Participant intros in breakoutrooms (Random assign first, then same for whole session)
- Daily schedule, learning objectives
- Code of conduct
- Chat (please use collaborative notes)

Participant preparations for each session

Copied from e-mail communication with participants before each session.

Session 1: In general you will only need your brain for the exercises and discussions, but some will have the option to go deeper, which may require some accounts or tools to be set-up. We will inform about these things the week before each session.

For Session 1 we recommend to have a GitHub account (https://coderefinery.github.io/installation/github/#github). There will be an optional exercise where you can build a lesson locally. This requires a basic Python environment (https://github.com/coderefinery/sphinx-lesson-template/blob/main/requirements.txt), but do not worry if you cannot or do not want to set up these, there will be plenty of other exercises.

Session 2: In general you will only need your brain for the exercises and discussions. To get the most out of this workshop day we recommend that you join the session with the same computer, display and microphone setup as you would usually use for teaching/ presenting. It may also help to get familar with the different options your computer and Zoom setup provides for screenshare and audio adjustments. But it is not a must.

Session 3: In general you will only need your brain for the exercises and discussions. If you want to share some "cool gem" in the last episode of this workshop day, please update your registration (link to edit can be found in the registration confirmation e-mail sent by support@coderefinery.org or noreply@neic.org). This is very low barrier, you do not have to be the developer of a tool or technique to share with the group what makes it useful for you; and it does not even have to be cool, sometimes the small "normal" things are the best. You can check out some topics that have been proposed so far in this GitHub issue (#90), you can also comment there if you want to add or change your topic. These will be lightning talk

length, so something between 2-5 min depending on how many are interested to share something, but we can collect them all in the collaborative notes for further exploration/reading. But it is not a must.

Session 4: This upcoming session (session number 4) will cover the technical aspects of CodeRefinery-style online teaching. These topics have never before been covered in this much detail, so if it's interesting to you, don't miss it. (You might also want to pass this on to others who are interested in technical production of online events; registration is still open!).

First, we talk about why we stream (how we got here, advantages, disadvantages, how to be an instructor in a livestreamed workshop, and what it looks like from a broad level from the production side)

Then, we show how we are able to release videos so quickly. This is useful far beyond CodeRefinery, and there are also some hands-on exercises. Come prepared to set up a Python environment and the ffmpeg command line tool if you want to try these.

Finally, we see details about how to set up Zoom→Livestream environment for yourself. There isn't time to do this as a proper exercise, but it's the starting point and the instructor will offer help after the course for those interested. If you want to try this out during the session, install OBS Studio in advance.