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.

Organizers and instructors:

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

Facilitators and instructors:

- TBD

Content and timing:

The workshop consists of four sessions every Tuesday between August 13th and September 3rd 2024, 9-12 CEST:

- Session 1: Contributing to and reusing CodeRefinery lesson materials (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.

Registration

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

Materials are work in progress

Materials will appear here before the workshop. You can find materials of previous similar trainings using the links below:

- First version of the course in 2019 and then in 2020: https://coderefinery.github.io/train-the-trainer/branch/instructor-training/
- Reworked material for our summer workshop 2022 and for the CarpentryCon 2022 workshop: https://coderefinery.github.io/train-the-trainer/branch/community-teaching/

Lesson template

Objectives

- Undertstand how the CodeRefinery lesson template is used to create new lessons

Instructor note

- Teaching: ? min
- Exercises: ? min

Here the episode sections and text ...

Keypoints

- Here we summarize keypoints.

Lesson development

Objectives

- Understand how collaborative development of lesson material works
- Be able to contribute to existing CodeRefinery lessons

Instructor note

- Teaching: ? min
- Exercises: ? min

Here the episode sections and text ...

Keypoints

- Here we summarize keypoints.

Post-workshop survey

Objectives

- Get to know the reasons and sources of inspiration behind major lesson and workshop updates

Instructor note

- Teaching: ? min
- Exercises: ? min

Here the episode sections and text ...

Keypoints

- Here we summarize keypoints.

Workshop overview and roles

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
- Understand the importance of onboarding and a welcoming community for volunteers in a workshop

Instructor note

- Teaching: ? min
- Exercises: ? min

Here the episode sections and text ...

Keypoints

- Here we summarize keypoints.

Onboarding

Objectives

- Here we will list learning objectives

Instructor note

- Teaching: ? min
- Exercises: ? min

Here the episode sections and text ...

Keypoints

- Here we summarize keypoints.

Co-teaching

Objectives

- Get to know the principle of co-teaching: How we do it and how you can too.

Instructor note

- Teaching: ? min
- Exercises: ? min

Why teach together?

It has been said **a lot**, especially in areas such as code development or scientific research, about the value of collaboration. Yet still today, the effort of teaching is made alone far too often: a person decides to share their knowledge (or gets assigned a study module) and starts building the actual teaching material basically from scratch. It seems much more logical, in the age of FAIR science and open knowledge, to release, develop, iterate, and maintain teaching material – including the contact sessions – **collaboratively** as well.

Ways to teach together

- Develop materials together - avoid duplication.
- Present the materials together ("proper" co-teaching, see Team teaching section on the CR manual).
- Use helpers extensively to tackle specific tasks commonly arising in online teaching process.
- Involve your learners too, e.g. using collaborative document (such as HackMD) for parallel and mass answers.

Advantages

- If you need to teach anyway, combined efforts take up less time.
- More engaging to the audience, taking some of the (sometimes daunting) expectation to "speak up" off of the students.
- Easier on-boarding of new instructors – one of them can be learning at the same time, either subtleties of the material or the teaching itself.
- Swiss-cheese principle: two "imperfect" teachers are **much** easier to find and complement each other than the extensively-prepared, absolute expert.

Challenges

- Additional effort needed of teacher and/or helper coordination – including syncing up their schedules!
- Materials might need to be (hopefully slightly) tuned to a specific target audience.
- Using simultaneous-teaching strategies is a learned skill, not identical to the classical lecturing.
- Online tools (HackMD, type-alongs) can potentially overload learners and teachers alike, if not used with care.

✍ Exercise

(What's better here – practical exercise or discussion?)

(TODO: Here goes the rest of the episode sections and text)

Keypoints

- Here we summarize keypoints.

Screenshare

Objectives

- Be aware of the importance of a well planned screenshare

Instructor note

- Teaching: ? min
- Exercises: ? min

Here the episode sections and text ...

Keypoints

- Here we summarize keypoints.

Sound

Objectives

- Be aware of the importance of a well balanced sound quality
- Test tips and tricks for achieving good sound quality

Instructor note

- Teaching: ? min
- Exercises: ? min

Here the episode sections and text ...

Keypoints

- Here we summarize keypoints.

Collaborative notes

Objectives

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

Instructor note

- Teaching: 25 min
- Questiongs & Answers: 5 min

Introduction

The Collaborative document is how you interact with the participants. The participants can ask questions and give feedback through the collborative document. During a CodeRefinery session there can be a large a volume of questions. A dedicated person, a HackMD-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 HackMD-manager.

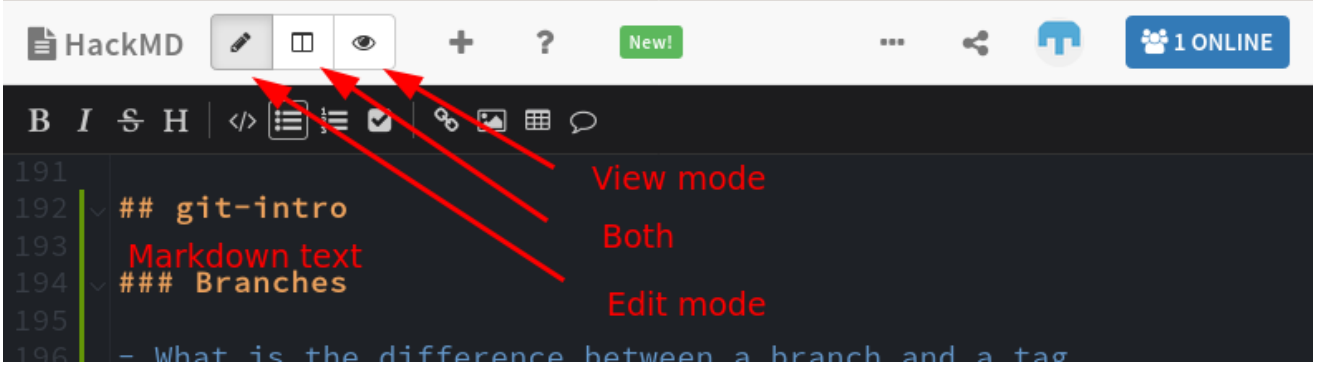
Collaborative document mechanics and controls

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



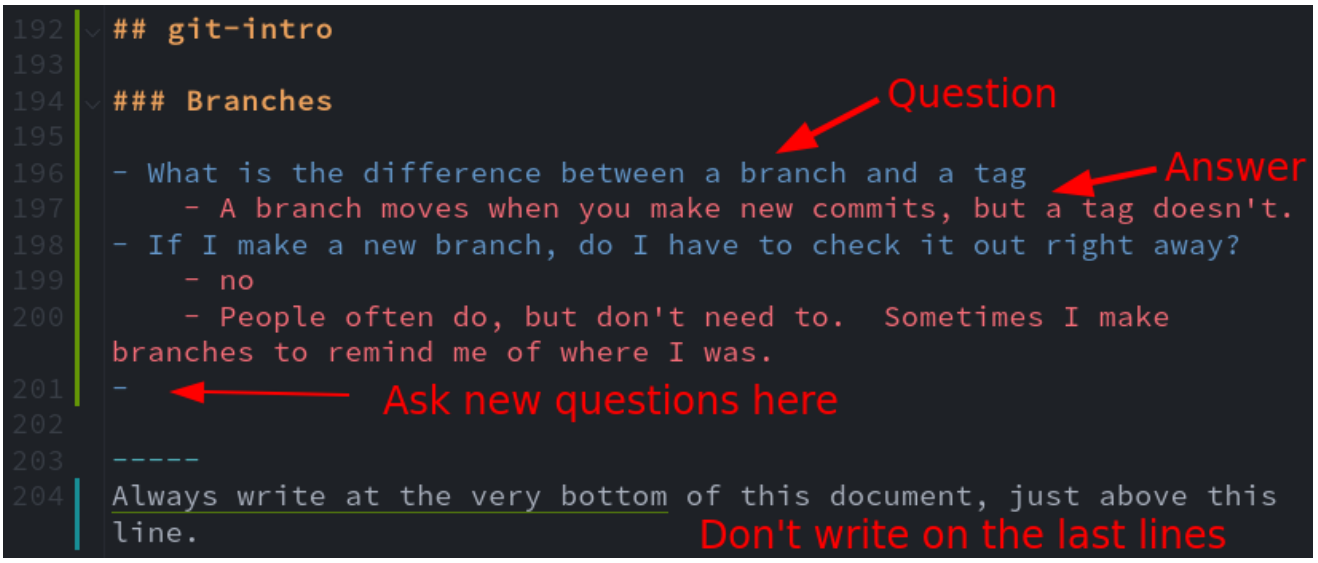
This may look slightly different on mobile devices and small windows.

- 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).

HackMD manager

We have one person who is a "HackMD 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 HackMD is updated with exercise, break, and other meta-information to keep people on track.

Below, (*) = important.

Before the workshop

- Create a new hackmd 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 hackmd (see below)

Most things to edit (everyone)

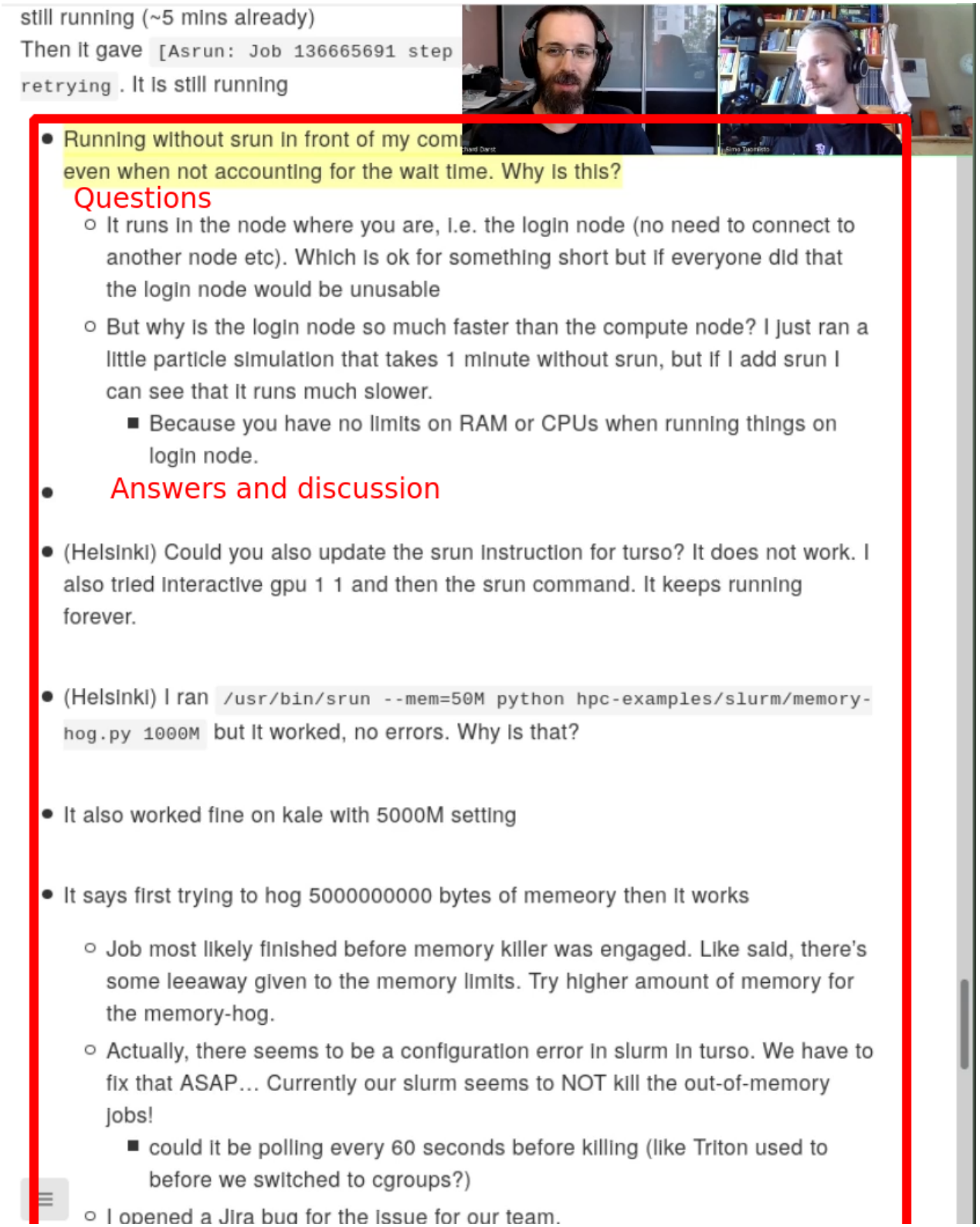
Make it easy to post after the course and consistent to follow:

- 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 HackMD 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.



A live demo of HackMD 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

- (*) 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 HackMD and answer.
- Anything important for following along should not be only said via voice. It needs to be in the HackMD, 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:

- During breaks and other times, share the HackMD (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:

- HackMD 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 HackMD to the workshop webpage (though others can do this, too).

HackMD 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 HackMD to website

HackMD 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

- Here we summarize keypoints.

Computational thinking

Objectives

- Get to know how the theory of computational thinking can be used in teaching

Instructor note

- Teaching: ? min
- Exercises: ? min

Here the episode sections and text ...

Keypoints

- Here we summarize keypoints.

Teaching philosophies

Objectives

- Get to know the teaching philosophies of CodeRefinery instructors

Instructor note

- Teaching: ? min
- Exercises: ? min

CodeRefinery teaching philosophies

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?

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

Recently 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 develop to few 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.

{: .challenge}

✍ 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.

Keypoints

- Here we summarize keypoints.

Cool gems

Objectives

- Here we will list learning objectives

Instructor note

- Teaching: ? min
- Exercises: ? min

Here the episode sections and text ...

Keypoints

- Here we summarize keypoints.

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.

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.

How did we get to the current state

- 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

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?

Q&A

Keypoints

- Streaming isn't that hard to understand
- There are benefits and disadvantages

Behind the stream

Objectives

- Get to know what happens "behind the stream" of a workshop
- See what the "broadcaster" sees and what they need to do.

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?

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?

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 *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

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

```yaml
- 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"
- Cross-platform, easy to use screencasting and streaming app.
- Real-time video mixer.

OBS user interface

- 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

Hardware requirements

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

CodeRefinery OBS configs

- https://github.com/coderefinery/obs-config

Installing the OBS config

Initial setup

Setup before each course

Keypoints

- 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.