CodeRefinery workshop software installation instructions

In CodeRefinery workshops, we will use and show a number of tools. We give you several options on how to participate (below).

- If you are using a university-managed computer from the Aalto University, please follow specific instructions.
- We recommend to ask a colleague/helper/team leader to go over the checklist with you.

What is the minimum of software/accounts I can get away with?

10 minutes

The bare minimum to participate in exercises for workshop part 1 (week 1: git and GitHub) and to only follow part 2 lectures/demonstrations:

- GitHub account

You will be able to participate in almost all exercises of part 1 (week 1) using the web interface of GitHub. This will hopefully give you a good understanding of the basics of Git and GitHub. For your own work after the workshop, you will probably want to use a local editor and Git on your computer.

I want to be able to do the exercises on my computer in an editor

20 minutes

We believe that **most learners will choose this option**. In this case we recommend two things:

- GitHub account
- Visual Studio Code (or if you prefer, other Text editors)
- Git in the terminal

I want to use Git from a terminal, not from an editor

15-60 minutes

In workshop part 2, I want to be able to run the exercises on my computer and have all tools available

30-90 minutes

In this case you want to use the command line interface for Git. This is good for power users but hard if you haven't used command lines before:

- GitHub account
- Git in the terminal
- SSH or HTTPS connection to GitHub from terminal
- See the command line crash course (video)

If part 2 happens over multiple days, we will do the exercises together. If part 2 is kept short within one week (week 2), we will not ask you to run the exercises during the workshop 2:

- GitHub account
- Visual Studio Code (or if you prefer, other Text editors)
- Conda environment

GitHub account

In CodeRefinery workshop, we use the public GitHub service and you need an account at https://github.com and a supported web browser. Basic GitHub accounts are free.

If you are concerned about the personal information to reveal to GitHub, for example how to keep your email address private, please review these instructions for keeping your email address private provided at GitHub.

We are trying to make it possible to follow this course also with other Git hosting services but this is work in progress.

Create a GitHub account

1. Go to https://github.com.
2. Click on the "Sign up" at the right-top corner.
3. Enter your username of your choice (if it is already used, you will get some suggestions), email address, and password.
4. Follow further instruction and verify your account.

GitHub may require you to enable multi-factor authentication (MFA). This is generally a good thing, but may take some time to set up. Luckily, you probably don't have to do this immediately. If you are prompted to enable MFA before the end of CodeRefinery, follow GitHub's instructions since they are usually pretty good.

How to verify that this worked

If you can log in to https://github.com, you should be good to go.

If you want to try a bit more, then try to create a new repository and then remove the repository again.

If you prefer not using GitHub

If you prefer not to create a GitHub account, you can still follow along with the lessons. We highly recommend the GitHub account if you are in a interactive workshop, because we can't support other platforms and a few exercises don't work. If you are following CodeRefinery at your own pace, you will have time to work these out.

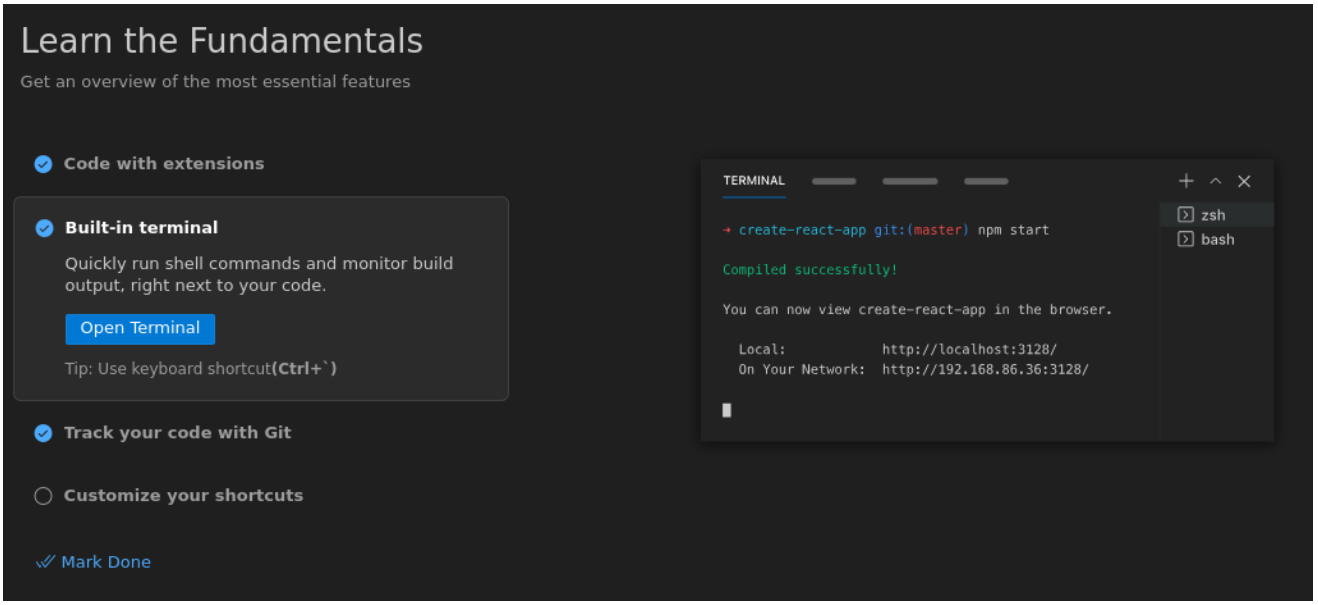
GitLab is a good alternative: https://gitlab.com or a local version.

- Anything can be done with other web repositories, but we don't say how. You will need to figure out where to click yourself.
- Most Collaborative Git lesson exercises won't work out of the box/will need adapting (but in principle the same idea exists with GitLab).
- The continuous integration (CI) lessons such as documentation and software testing won't work without adapting. With GitLab you need to provide your own CI runners, which is probably beyond the scope of what you would do just for this workshop.

Visual Studio Code

Visual Studio Code is a popular source code editor. It is **free and open source** and runs on your desktop and is available for Windows, macOS, and Linux. It has a rich ecosystem of extensions for languages such as C++, Fortran, R, C#, Matlab, Java, Python, PHP, and Go. **If you are new to text editors, we recommend to start with this one**.

- Please visit the download page for installation instructions. But please note that Visual Studio Code sends data to Microsoft.
- Please also browse the page about Using Git source control in VS Code.
- VSCodium is the same software without Microsoft tracking.
- Windows (and any operating system that has the option): when installing, select the option "Add to PATH" (this is default).



When installing Visual Studio Code, I have selected the options "Built-in terminal" and "Track your code with Git".

Using VS Code as a git editor

This will set VS Code as the editor that Git starts. It will start a new tab, and Git will wait until you save and close that tab. Git for Windows on Windows may automatically set this if you select it as an editor:

```
$ git config --global core.editor "code --wait"
```

Text editors

A **text editor** edits plain text, and is important for almost any type of computer work.

If you are new to text editors

If you are new to text editors, we recommend that you practice a bit: try to create a file, edit it, save it, re-open it.

There are text editors with a graphical user interface and text editors that allow you to edit text in the terminal, for example on a remote cluster (and some allow both modes of operation). This page tells you how to connect different editors to git, so that command line git will open them.

Demonstration. Command line git starting the VS Code graphical editor to write a commit message (don't worry about what the command does, we cover this in the CodeRefinery workshop). Picture should be animated.

Choosing the right editor is a matter of preferences. Since we often spend significant portions of our days editing text and source code, it can be valuable to invest time into learning your favourite editor really well. Below we list few common options and give some pros/cons.

Visual Studio Code (graphical)

If you are new to text editors, we recommend to start with this one. We have an extra page for it: Visual Studio Code.

If you choose a different editor

If you choose a different editor, make sure to browse its documentation on how to conntect it to Git.

This command will configure git to start VS Code as its editor. (This will happen automatically if you select the VS Code option when installing Git for Windows.)

```
$ git config --global core.editor "code --wait"
```

Nano (terminal)

Easy to start but comes with minimal functionality.

Windows | macOS | Linux

Nano is installed as part of the Git for Windows installer and no extra installation is needed. It is available from the git-bash shell.

To set it as the default editor for Git:

```
$ git config --global core.editor nano
```

The keyboard shortcuts are displayed at the bottom of the editor window. Using this shortcuts involves pressing and holding down the control key (Ctrl) on your keyboard and pressing another key. The pressing down and holding the Ctrl key is represented by a hat "^".

Tutorial: https://www.tutorialspoint.com/how-to-use-nano-text-editor

To create or open a file called mytext.txt:

```
$ nano mytext.txt
```

To save content: Ctrl + o (hold the Ctrl key and press the o)

To close a file: Ctrl + x (hold the Ctrl key and press the x)

To move up, down, left or right with the document: Use the arrow keys and Page-up, Page-down keys

Deleting text: Navigate to where the text to be deleted located in the document using arrow keys. Use the Delete or Backspace keys to delete text.

To find: Ctrl + w then type the word to find and press enter (please note it is w not f as in most other editors).

Notepad++ (graphical)

If you are on Windows and want to use Notepad or Notepad++, you can configure this by providing the full path to the executable and optionally set some options. For example (adjust the path if needed, and note the quotation):

```
$ git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' -
multiInst -notabbar -nosession -noPlugin"
```

Vi/Vim

This editor takes some effort to get started. But has more functionality than Nano, especially if you write programming code. Syntax highlighting, clever copy-paste and better refactoring are some features.

Interactive Vim tutorial: http://www.openvim.com

To create or open a file called mytext.txt:

```
$ vi mytext.txt
```

To close a file: Hit Escape, then type `:wq` and hit Enter.

Emacs

Like Vim, Emacs takes some effort and learning to get started and offers almost unlimited functionality. It is possible to interact with version control, even compile and run code, send emails, etc. all from the editor itself.

Emacs guided tour: https://www.gnu.org/software/emacs/tour/

To create or open a file called mytext.txt:

```
$ emacs mytext.txt
```

To close a file: Type Ctrl-x followed by Ctrl-c.

Git in the terminal

This page is for setting up Git **if you want to use it in a terminal**.

If you have never been in a terminal shell, please briefly read this crash course, and consider watching this shell tutorial video by Richard Darst (20 min).

Git is a version control system that lets you track who made changes to what and when, and has options for updating a shared or public version of your code on source code repository hosting platforms such as GitHub or GitLab or Bitbucket.

Please follow the installation, configuration, and verification instructions below.

Installation

Windows | macOS | Linux

You can follow the steps below by watching the Carpentries video tutorial.

- Download the Git for Windows installer.
- Run the installer and follow the steps below:

  1. Click on “Next” four times (two times if you’ve previously installed Git). You don’t need to change anything in the Information, location, components, and start menu screens.

2. **From the dropdown menu select an editor**:

    - If you use VS Code: Select that from the list. You must have VS Code installed already.
    - If you don't know what else to do: "Use the Nano editor by default" (NOTE: you will need to scroll up to find it) and click on "Next".**
    - If are comfortable with another editor and know it is installed, you can select it - but Nano is the safest if you don't know what to choose.

    After this, you don't have to do step 2 "selecting an editor" step below (but that step lets you change your choice).

3. **On the page that says "Adjusting the name of the initial branch in new repositories", choose "Override the default branch name for new repositories" and choose `main`. This will ensure the highest level of compatibility for our lessons.**
4. Ensure that "Git from the command line and also from 3rd-party software" is selected and click on "Next". (If you don't do this Git Bash will not work properly, requiring you to remove the Git Bash installation, re-run the installer and to select the "Git from the command line and also from 3rd-party software" option.)
5. Choosing the OpenSSH executable: "Use bundled OpenSSH" and click "Next".
6. **Ensure that "Use the native Windows Secure Channel Library" is selected and click on "Next".**
7. Ensure that "Checkout Windows-style, commit Unix-style line endings" is selected and click on "Next".
8. **Ensure that "Use Windows' default console window" is selected and click on "Next".**
9. Ensure that "Default (fast-forward or merge) is selected and click "Next"
10. Ensure that "Git Credential Manager" is selected and click on "Next". (if "Core" is an option, pick that)
11. Ensure that "Enable file system caching" is selected and click on "Next".
12. Configuring experimental options: don't select either and click "Next".
13. Click on "Install".
14. Click on "Finish" or "Next".

- If your "HOME" environment variable is not set (or you don't know what this is):

    1. Open command prompt (Open Start Menu then type `cmd` and press enter)
    2. Type the following line into the command prompt window exactly as shown: `setx HOME "%USERPROFILE%"`
    3. Press enter, you should see `SUCCESS: Specified value was saved.`
    4. Quit command prompt by typing `exit` then pressing enter.

- If you are using a Windows laptop managed by your organization, it might be that your "HOME" variable is pointing to a network disk (a storage location that is not physically in your laptop, usually starting with "Z:"). This should not be an issue per se, but it has caused problems to some participants in the past. If you are

experiencing issues with Git Bash, you can try setting HOME to a location on the physical disk, for example “C:\Users$USERNAME”. Please note that, depending on the settings of your organisation, automatic backups might not cover the newly set home folder. Follow instructions from your local ITs on how to make sure that also those folders are backed up.

This will provide you with both Git and Bash in the Git Bash program. You can then start it by searching for “Git Bash” in your Start menu.

Text copied and adapted from: the Carpentries set up page

Configuration

⚙ Prerequisites

We assume that you have:

- Bash (or Zsh on macOS) shell terminal on your machine
- Git installed on your machine
- Nano editor available (unless you prefer a different terminal editor)

Video instruction

This video (9 min) shows how to configure Git. If you experience problems, watch this troubleshooting video.

Step 0: Start your terminal that has git

Windows | macOS | Linux

Start the application Git Bash.

This is the command line interface. Did you read/watch the crash course above?

Step 1: Setting user name and email

First, the following commands will set your user name and email address:

```
$ git config --global user.name "Your Name"
$ git config --global user.email yourname@example.com
```

If you are unsure which email to use, use the one that you have registered at GitHub with.

This name and email will be public on GitHub when you make contributions, including in this course. If you prefer to not use a real email address but still want to make sure that GitHub counts your contributions, you can use `yourusername@users.noreply.github.com` (replace `yourusername`).

This is important since your Git commits use this information. The `--global` option uses this information for every repository for this user on this computer.

Different name/emails for different repositories

If you need different name/email for different repositories see this StackOverflow question for configuring based on directory and the raw configuration options, including something that works based on remote URLs (but it needs a newer git version).

Step 2: Setting an editor

It is important to set also the default text editor to use with Git. **We recommend to use nano if you do not have any other preferences** (it is included and simple for demonstrations). But you can instead use your preferred editor (VS Code, vim, emacs or any other editor of your choice, see Text editors for more instructions):

This sets the editor to Nano:

```
$ git config --global core.editor nano
```

To set other editors, see Text editors.

Step 3: Set default branch

The default branch on GitHub and GitLab is `main` but it used to be `master`. In our workshop we will use `main` as the default branch also in the terminal to be consistent with GitHub and GitLab:

```
$ git config --global init.defaultbranch main
```

This configuration will unfortunately have no effect on Git older than 2.28. **But this is not a problem**. During the lesson we will have a workaround for this.

Step 4: Two more settings for Windows only

We have found that these two settings prevent warnings and/or seeing no output in some commands on Windows (no need to run these on macOS or Linux). At the bottom of this page we motivate in more detail what these two do:

```
$ git config --global core.autocrlf false
$ git config --global core.pager cat
```

Verification

⚙ Prerequisites

We assume that you have:

- Bash (or Zsh on macOS) shell terminal on your machine
- Git installed and configured, with `nano` as its editor on your machine (unless you prefer a different terminal text editor)

The following shows the same steps that are shown in the video above. You can see verification part in the same video for configuration at 5:50.

Step 1: Check your Git version

Check your git version:

```
$ git --version

git version 2.39.2
```

- We recommend at least 2.23.
- Ideal is 2.28 and newer.
- But the lesson and workshop will still work even with Git 2.0 and we will add instructions/workarounds for Git below 2.28 or 2.23.

Step 2: `git init -b main` inside a new folder

Create a new example folder, step into it, and initialize a repository. Do **not** `git init -b main` in your home directory:

```
$ mkdir example
$ cd example
$ git init -b main
```

Step 3: Make a new file

Make a new file with some content:

```
$ nano example.txt
```

Write some text in the `example.txt`, save (`ctrl+O` then `ENTER`) and exit (`ctrl+X`).

Step 4: Stage the change

Stage a newly made file `example.txt` before committing the change and record in Git:

```
$ git add example.txt
```

Step 5: Commit the change with a message

The following command will commit the change. It should open the editor which you have configured. Then, add a commit message such as `initial commit` at the very top. Then save an exit (if you use the default `nano` editor, `ctrl+O` `ENTER` then `ctrl+X`):

```
$ git commit
```

Step 6: See the change log

```
$ git log
```

If you see now something line this your Git is configured for the workshop (the name, email, commit message and also the "12e4cb8..." part will be different in your case):

```
commit 12e4cb892140bd14a413895b3b36c27db198eb22 (HEAD -> main)
Author: Radovan Bast <bast@users.noreply.github.com>
Date:   Fri May 15 16:41:13 2020 +0200

    making a test commit
```

Troubleshooting

Where is this configuration stored?

To see where this information is stored (`--show-origin` works on git version 2.8 or greater only), use:

```
$ git config --list --show-origin
```

On Windows `git log`, `git diff`, `git branch` or other git commands show no output at all

It seems this can be fixed by these configuration settings:

```
$ git config --global pager.log off
$ git config --global pager.diff off
$ git config --global pager.show off
$ git config --global pager.config off
$ git config --global pager.stash off
$ git config --global pager.help off
$ git config --global pager.blame off
$ git config --global pager.branch off
$ git config --global pager.annotate off
```

Following other manuals and documentation, it seems that all the above can be set with the following (`cat` should be available within Git Bash but also PowerShell):

```
$ git config --global core.pager cat
```

On Windows you see the warning “LF will be replaced by CRLF the next time Git touches it”

Solution is to set this:

```
$ git config --global core.autocrlf false
```

SSH or HTTPS connection to GitHub from terminal

Why are we doing this?

We need to connect to GitHub (or whatever other repository we may be using), and for that to work, we have to be able to identify ourselves. This is actually a pretty hard thing to do, and there are two main options **when authenticating from a terminal**.

You do not need to configure the below if you are using Visual Studio Code.

What are the options?

SSH | HTTPS | VS Code

This is recommend if you are on Linux or macOS (it does work on Windows, but since the HTTPS method is included by default, you may as well use that instead).

Secure Shell (SSH) is the standard program for connecting to remote servers. It's well worth learning anyway.

These days, every major operating system has it built in, but it requires more set up using "SSH keys" to use it for GitHub. It's really useful to set this up and get to know it well, even for things other than git.

You should remember your choce for the lessons.

Instructions

Based on your decision above, try to set it up. You can try the other one if your first choice doesn't work. There's nothing wrong with doing both of them.

SSH | HTTPS | VS Code

Watch this in video form

This CodeRefinery video about ssh keys (7 min) shows how to set up SSH connection to GitHub. If you are new to SSH keys, We recommend you watch this first to understand what is going on.

For the instructions, please follow this guide from GitHub to connect to GitHub with SSH keys. These same instructions work with services other than GitHub as well (except the adding the key to GitHub part).

- If you think you might already have SSH keys, start with Checking for existing SSH keys
- Then start with Generating a new SSH key and adding it to the ssh-agent
- Then Adding a new SSH key to your GitHub account
- Then Testing your SSH connection (note that this is the same as the verification steps below)
- If on Windows, Working with SSH key passphrases is optional but can save you from retyping the passphrase so many times.

How to verify that it worked

SSH | HTTPS | VS Code

Try this in your terminal shell:

```
$ ssh -T git@github.com
```

If you set up ssh keys correctly, you will see:

```
Hi yourusername! You've successfully authenticated, but GitHub does not provide
shell access.
```

If it says **"You've successfully authenticated"** then it works and your SSH keys are properly set up with GitHub.

You possibly see this warning (the IP and the fingerprint may look differently):

```
$ ssh -T git@github.com

The authenticity of host 'github.com (140.82.121.4)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

Before typing "yes", verify that the fingerprint is one of these.

What this means: SSH is a secure protocol to send data between two computers but the very first time you ever connect to the remote host, SSH asks if this is really the host/server we meant to talk to and to verify that nobody is trying to impersonate the other host. Once we validate the connection, SSH will remember that we trust that host and not ask this question again.

Conda environment

[this page is adapted from https://aaltoscicomp.github.io/python-for-scicomp/installation/]

You only need this if you wish to go through examples and exercises during the second workshop week on your own computer.

Choosing an installation method

For this course we will install an **isolated environment** with following dependencies:

environment.yml | requirements.txt

```yaml
name: coderefinery
channels:
  - conda-forge
  - bioconda
dependencies:
  - python>=3.10,<3.14
  - click=8.3.1
  - ipywidgets=8.1.8
  - jupyterlab=4.5.3
  - jupyterlab-git=0.51.4
  - matplotlib=3.10.8
  - myst-parser=5.0.0
  - nbdime=4.0.2
  - numpy=2.4.1
  - pandas=3.0.0
  - pytest=9.0.2
  - pytest-cov=7.0.0
  - seaborn=0.13.2
  - snakemake-minimal=9.14.8
  - sphinx=8.2.3
  - sphinx-autobuild=2025.8.25
  - sphinx_rtd_theme=3.1.0
  - sphinx-autodoc2=0.5.0
```

If you have a tool/method that works for you, keep using it

If you are used to installing Conda packages and know what to do with the above files, please follow your own preferred installation method.

If you are new to Conda or unsure how to create isolated environments from files above, please follow the instructions below where we do this with the help of Miniforge.

If you already use **Anaconda** to manage your software environments, please continue using it. In this case you can skip Installing Miniforge and go directly to Installing the software environment.

Why do we recommend Miniforge and how does it relate to Conda?

There are many choices and we try to suggest a good compromise There are very many ways to install packages with pros and cons and in addition there are several operating systems with their own quirks. This can be a huge challenge for beginners to navigate. It can also difficult for instructors to give recommendations for something which will work everywhere and which everybody will like.

Below we will recommend **Miniforge** since it is free, open source, general, available on all operating systems, and provides a good basis for reproducible environments. However, it does not provide a graphical user interface during installation.

But Miniforge is only one option. Unfortunately there are many options and a lot of jargon. Here is a crash course:

- **Conda** is a package manager: it allows distributing and installing packages, and is designed for complex scientific code.
- **Mamba** is a re-implementation of Conda to be much faster with resolving dependencies and installing things.
- An **environment** is a self-contained collections of packages which can be installed separately from others. They are used so each project can install what it needs without affecting others.
- **Anaconda** is a commercial distribution of Python+Conda+many packages that all work together. It used to be freely usable for research, but since ~2023-2024 it's more limited. Thus, we don't recommend it (even though it has a nice graphical user interface).
- **conda-forge** is another channel of distributing packages that is maintained by the community, and thus can be used by anyone. (Anaconda's parent company also hosts conda-forge packages)
- **Miniforge** is the installer recommended by the conda-forge community. It operates via the command line.
- **Miniconda** is a distribution of conda pre-configured to use the Anaconda channels.

Installing Miniforge

Download the latest release from the Miniforge download overview. This installs the base, and from here other packages can be installed.

Unsure what to download and what to do with it?

Windows | MacOS | Linux

You want to download and run `Miniforge3-Windows-x86_64.exe`.

When viewing the "Advanced installation options" step of the installer, leave the "Create shortcuts (supported packages only)." option **checked**, otherwise it will be difficult to launch the Miniforge prompt later.

Installing the software environment

First we will start a command line environment in a way that activates conda/mamba. Then we will install the software environment from this environment.yml file (this is the same file as on top of this page).

An **environment** is a self-contained set of extra libraries - different projects can use different environments to not interfere with each other. This environment will have all of the software needed for this particular course.

We will call the environment `coderefinery`.

Windows | Linux / MacOS

Use the "Miniforge Prompt" to start Miniforge. This will set up everything so that `conda` and `mamba` are available. Then type (without the `$`):

```
$ mamba env create -n coderefinery -f
https://raw.githubusercontent.com/coderefinery/software/main/environment.yml
```

Activating the software environment

Windows | Linux / MacOS

Start the Miniforge Prompt. Then type (without the `$`):

```
$ conda activate coderefinery
$ jupyter-lab
```

How to verify the environment

Make sure you are in the Miniforge prompt (Windows) or in a terminal (Linux/MacOS).

The following command should return a version number and not an error message:

```
$ conda --version
```

Is the coderefinery environment installed?

```
$ conda env list
```

Make sure it's activated (Activating the software environment). Once activated, try the following commands.

You should see versions printed and not see errors (exact version numbers are not too important):

```
$ jupyter-lab --version

4.1.2
```

```
$ pytest --version

8.0.2
```

```
$ sphinx-build --version

sphinx-build 7.2.6
```

```
$ snakemake --version

8.5.3
```

Removing the software environment

Windows | Linux / MacOS

In the Miniforge Prompt, type (without the `$`):

```
$ conda env list
$ conda env remove --name coderefinery
$ conda env list
```

If you encounter any problems

- Ask a friend/colleague/local IT support. These are standard tools which many people can help with (and don't be afraid to ask - it's an unfortunate fact that software installation and configuration is hard).
- Before a workshop we offer 2 drop-in installation-help sessions. Please try by yourself first, but if you cannot solve problems, join in the sessions. Schedule is shown in the workshop website.
- Ask in the `#help` stream on the CodeRefinery chat. You can also send an email to support@coderefinery.org to ask questions. Please describe the problem concretely as well as your operating system.

Visual diff tools (optional)

Visual diff tools allow to visualize differences between files side by side. They can be used independently of version control but when coupled with Git, they can provide a nice alternative to command line output to show differences between file versions or branches.

Windows | macOS | Linux

On Windows we recommend the program meld. Please follow installation instructions from http://meldmerge.org.

This is how you can configure Git to open Meld when doing `git difftool`:

```
$ git config --global diff.tool meld
$ git config --global mergetool.meld.path "C:\Program Files (x86)\Meld\Meld.exe"
```

For meld version 3.20.3, because of reasons discussed here, you can have multiple "Could not create key" errors that you can get around by clicking 'Ignore'. Then, to open Meld when doing `git difftool`:

```
$ git config --global diff.tool meld
$ git config --global mergetool.meld.path "$LOCALAPPDATA\Programs\Meld\Meld.exe"
```

How to verify the installation

To test it create two text files which are similar and then compare them with Meld or Diffuse or Opendiff (or the tool which you have chosen above for visual differences):

```
$ meld file1 file2
```

or:

```
$ diffuse file1 file2
```

or:

```
$ opendiff file1 file2
```

You should now see both versions side by side.

To test launching Meld through Git, move into a folder that contains two or more files and instruct Git to compare files. For example, to see the difference between `file1.txt` and `file2.txt`:

```
$ git difftool --no-index file1.txt file2.txt
```

Why are we asking participants to install software?

We could provide a virtual machine in the cloud with everything pre-installed and ask participants to connect to it.

However, we prefer that participants follow the workshop in the environment where they are likely to work in future, and know how to set this up and configure it. We would like that participants are able to continue using tools also after the workshop, in a familiar development environment of their choice.

We are aware that this choice comes at a price: it is very difficult for us to provide installation instructions that work "everywhere" and it requires from participants to do some work before they arrive at a workshop. **Yes, installing software is harder than you might expect.** Don't be afraid to ask for help in setting it up: this is also a good skill to have.

We are also aware that not everybody is comfortable with creating new accounts for this workshop. And therefore we provide Instructions for removing a GitHub repository or account, if you wish to do so.

Instructions for removing a GitHub repository or account

You may wish to remove the projects and accounts that we have created. Here we list instructions for how to achieve that:

- Deleting a GitHub repository
- Deleting a GitHub account

Credit and license

These installation instructions are inspired by and derived from work by Software Carpentry which is licensed under the Creative Commons Attribution license (CC BY 4.0).

In particular, we have copied a number of instructions from the workshop template website (instructions to install Bash and Git), as well as copied and adapted most of their license text.

Instructional Material

All CodeRefinery instructional material is made available under the Creative Commons Attribution license. The following is a human-readable summary of (and not a substitute for) the full legal text of the CC BY 4.0 license.

You are free:

- to **Share** - copy and redistribute the material in any medium or format
- to **Adapt** - remix, transform, and build upon the material

for any purpose, even commercially. The licensor cannot revoke these freedoms as long as you follow these license terms:

- **Attribution** - You must give appropriate credit (mentioning that your work is derived from work that is Copyright (c) CodeRefinery and, where practical, linking to http://coderefinery.org), provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

No additional restrictions - You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits. With the understanding that:

- You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.
- No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

Software

Except where otherwise noted, the example programs and other software provided by CodeRefinery are made available under the OSI-approved MIT license:

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.