Lab Handbook

Bram Kuijper

2022-06-16

Contents

Chapter 1

Introduction

This document gives you an overview of some of the key computing practices used in our lab. Enjoy!

Chapter 2

Installing and using software to do your work

To do your research in theoretical or computational biology, you will need to install a bunch of software. I typically try to keep things as free and open access as possible, so that you don't have hassle with license fees. However, this is not always possible (e.g., research projects involving Mathematica or matlab).

2.1 Finding out what software you need

Don't install *all* the software listed below. Rather contact me to discuss what software you need. Typically, you need one (or two) of the following options:

1. The Rstudio environment
2. A Python environment
3. A C++ environment
4. A Mathematica environment

2.2 The Rstudio environment

R is a very commonly used programming environment and particularly useful for data analysis. Its disadvantages are that it is slow and that the programming language is arcane, to say the least. All pros and cons considered, if your project is data heavy, R is a great choice.

Rather than using the absolute bare bones version of the programming environment R, we use Rstudio, because it provides a full-blown IDE (integrated development environment) with text editor, variable inspector, file browswer and more. You can either use your web browser to go to the online Rstudio server that is offered by the University of Exeter. Alternatively, you can choose

to use a locally installed version of Rstudio (see below for installation instructions). If you don't know whether to choose the server-based version or the local installed he server-based version of Rstudio is probably the easiest to use, as it already has tons of packages pre-installed. However, if you have no uni access anymore or don't have continuous internet access, a locally installed version of Rstudio is best.

2.2.1 Installing Rstudio on a mac

A great way of keeping Rstudio up-to-date is to first install the Homebrew package manager Homebrew. This is a small program that allows you to install and update multiple programs on your Mac. Indeed, almost any research software that you will need on your Mac can be installed with homebrew. Using homebrew has the advantage that installation involves a single command, rather than you having to search for the correct version of the software online. Moreover, all installed packages will be updated to their latest versions by using two simple commands: `brew update`, following by `brew upgrade`.

2.2.2 Installing homebrew

To install homebrew, you need to open the Terminal app (see here about where to find the Terminal app on your Mac). Once the terminal is open, copy the single-line install statement from the Homebrew website and paste it into the Terminal app. Then press 'Enter'. You will get a bunch of straightforward questions, after which homebrew installs itself.

2.2.3 Installing your first application using homebrew

After homebrew is installed, you can use it to install other applications. You can search for software available for install through homebrew by using the website formulae.brew.sh.

Let us install the [Textmate] text editor that we might need later. We do so using the Terminal app app, in which we type

```
brew install --cask textmate
```

2.2.4 Updating all applications that have been installed via homebrew

This is where things get handier than installing all software individually:

```
brew update && brew upgrade
```

2.3 Rstudio

2.3.1 Using the web-based Rstudio server

You can simply access this by using your web browser. You can access the Rstudio server here: https://rstudio01.cles.ex.ac.uk, using your University of Exeter login.

Once logged in, make sure to run the 4.x.x version, rather than a 3.x.x version. You can change versions on the top right of your Rstudio window within in your web browser. See the image below

2.3.2 Installing Rstudio on your own computer on windows or mac

If you have a Windows machine, the best way to install Rstudio is to download the Desktop version. If you are on a mac, one could use:

```
brew install --cask rstudio
```

2.4 Installing Python

For those of you who will need to install python (installing python is more the exception than the norm, hence ask if unsure), it is best to install the Anaconda distribution, which provides the whole python bundle and comes with a package manager.

2.5 Windows: Accessing remote servers using mobaxterm

For some projects we need to access a remote server to run our software. For this we use the programme mobaxterm.

To install mobaxterm, watch this handy video. Make sure you download the 'Portable version' rather than the 'Installer' version.

The mobaxterm-xxx.exe file can be ran from anywhere on your operating system (hence, the name Portable). Hence, try to put in a location where you will not forget it. Alternatively make a shortcut to the mobaxterm-xxx.exe file (Right mouse click > create shortcut) and then drag the shortcut to the taskbar in Windows. See this video of some dude explaining everything taskbar-related on windows.

2.5.1 Using Mobaxterm to access a remote location

Now let's use mobaxterm to access the remote server. ## Installing software to develop C++ programs {#section:installcpp} When working with the pro-

gramming language C++, we will need to use the following software:

1. A UNIX Terminal (already installed on a mac, you will need to install on windows - see below)
2. A C++ compiler, which in our case will be `clang` on mac and `g++` on windows (see below).
3. A better-than-normal text editor. When on Windows, install the freely available text editor Notepad++. See the Notepad++ website for installation instructions. When on a mac, install an editor such as textmate (see below).
4. A software build environment like make or cmake.

2.6 Yes, we really need a UNIX terminal.

You will need a UNIX terminal to access a broad collection of tools, namely the compiler `g++` o turn your C++ code into an executable programme, the command `git` to get code from repositories and `make` or `cmake` to automate the building of your code.

Sure, it would be possible to avoid a UNIX toolchain, by using Microsoft's Visual Studio or another compiler. However, chances are that we'll be running things on one of the University's Linux computing clusters later on. If one has been working with Visual Studio, it typically is a massive pain to switch back to the different tools and compilers used on these Linux clusters, particularly when we would use associated libraries such as `gsl` or `eigen`. By contrast, when you already use a UNIX terminal on your local computer, moving to use the University's Linux cluster is very easy.

2.7 Windows: install MSYS2 as our UNIX terminal

To get a UNIX terminal running on windows, we will install the MSYS2 environment. See here for the installation video. There is also another video that shows you how to subsequently work with the compiler, once installed.

2.7.1 Accessing MSYS2 once installed

Once everything is installed, use the MSYS2 MinGW x64 executable, rather than the default MSYS2 executable. To start this, simply type `msys` in the windows search box (see screenshot) and the `x64` version will appear in the menu:

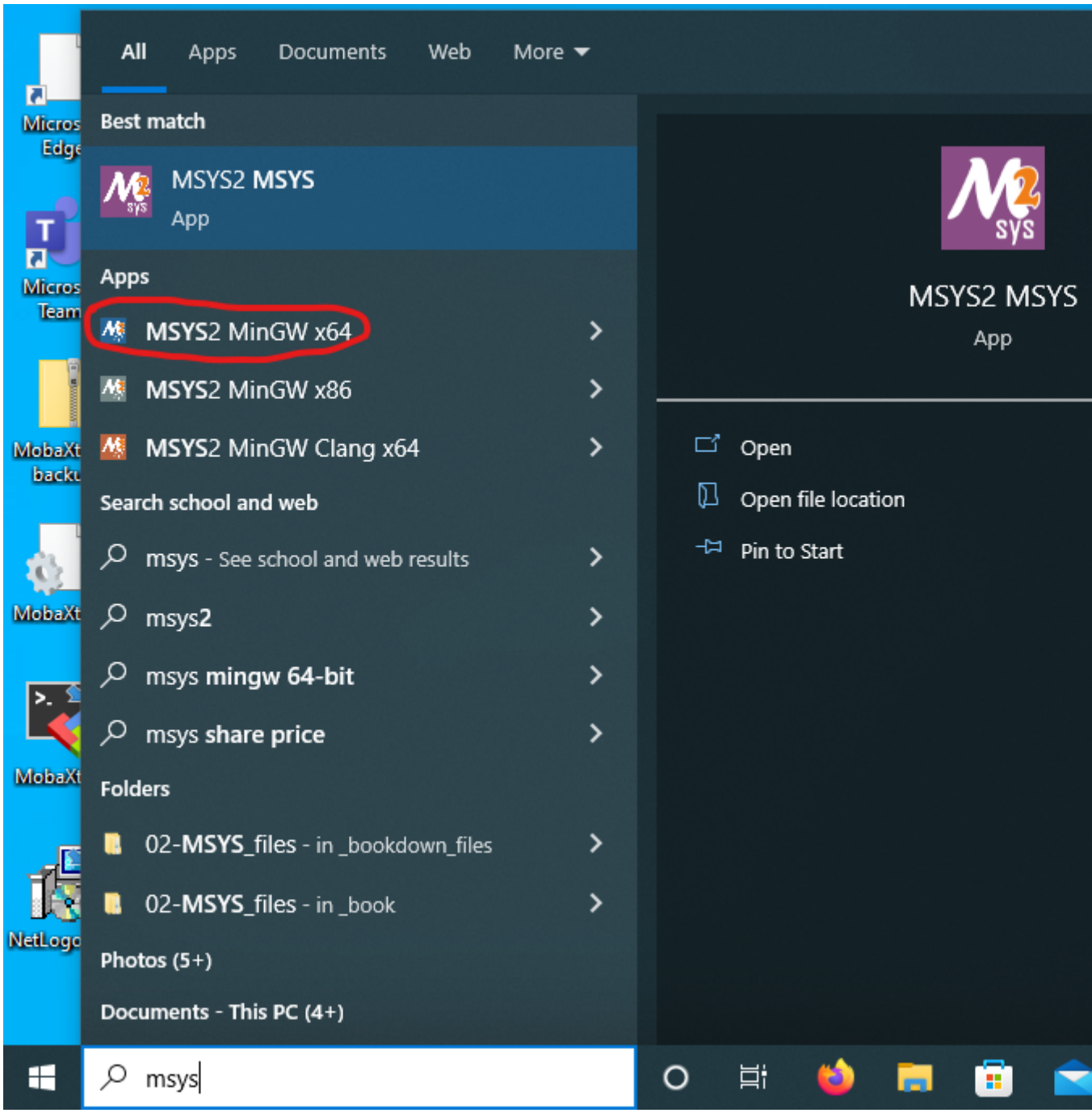

Figure 2.1: Location of MSYS x64 in the windows menu. The default MSYS2 app will appear on top, but the x64 version can be found right below (red circle).

2.7.2 Accessing MSYS2's home directory through Windows File Explorer

The home directory in MSYS2 is indicated by the `~` sign, which points to a directory on your hard drive. However, the MSYS2 home directory is not the same as your Windows home directory! Rather, MSYS2's home directory `~` maps to something like `C:/msys64/home/$your_user_name`, rather than to the usual location of the windows home directory which is something like `C:/Users/$your_user_name/home`, where you need to replace `$your_user_name` with your windows user name.

Knowing that `~` is somewhere else than the standard home directory, you can still use the Windows File Explorer to access this location and inspect the files present in `~`. Click the `This PC` in the left column of the Windows File Explorer and go to `C:/msys64/home/$your_user_name`. Here an example screenshot to clarify matters:

2.7.3 Installing a command-line C++ compiler within MSYS2

Rather than using any C++ compiler like Visual Studio or Code Blocks, we use the compiler g++, as that compiler is the same as used on our computing clusters. To install it, see the instruction video here.

2.7.4 Installing `git` within MSYS2

We need `git` to download and synchronize code repositories. To install `git` within MSYS2 type the command

```
pacman -S git
```

2.7.5 Windows: use notepad++ as your text editor of choice

To edit and inspect source files, please **do not** use the standard text editor on windows, which is notepad. This crippled and lame editor cannot read source files with line endings that are used on different operating systems like linux or mac. Moreover, it does not have syntax colouring, which is a life saver when reading code.

Hence, please install the free editor Notepad++ which provides syntax colouring and more. You can download a copy of Notepad++ here.

If you don't like Notepad++, have a google yourself. Any code editor that supports syntax coloring of C++,R,Python files and supports `CR/LF` and `LF` line endings should be OK.

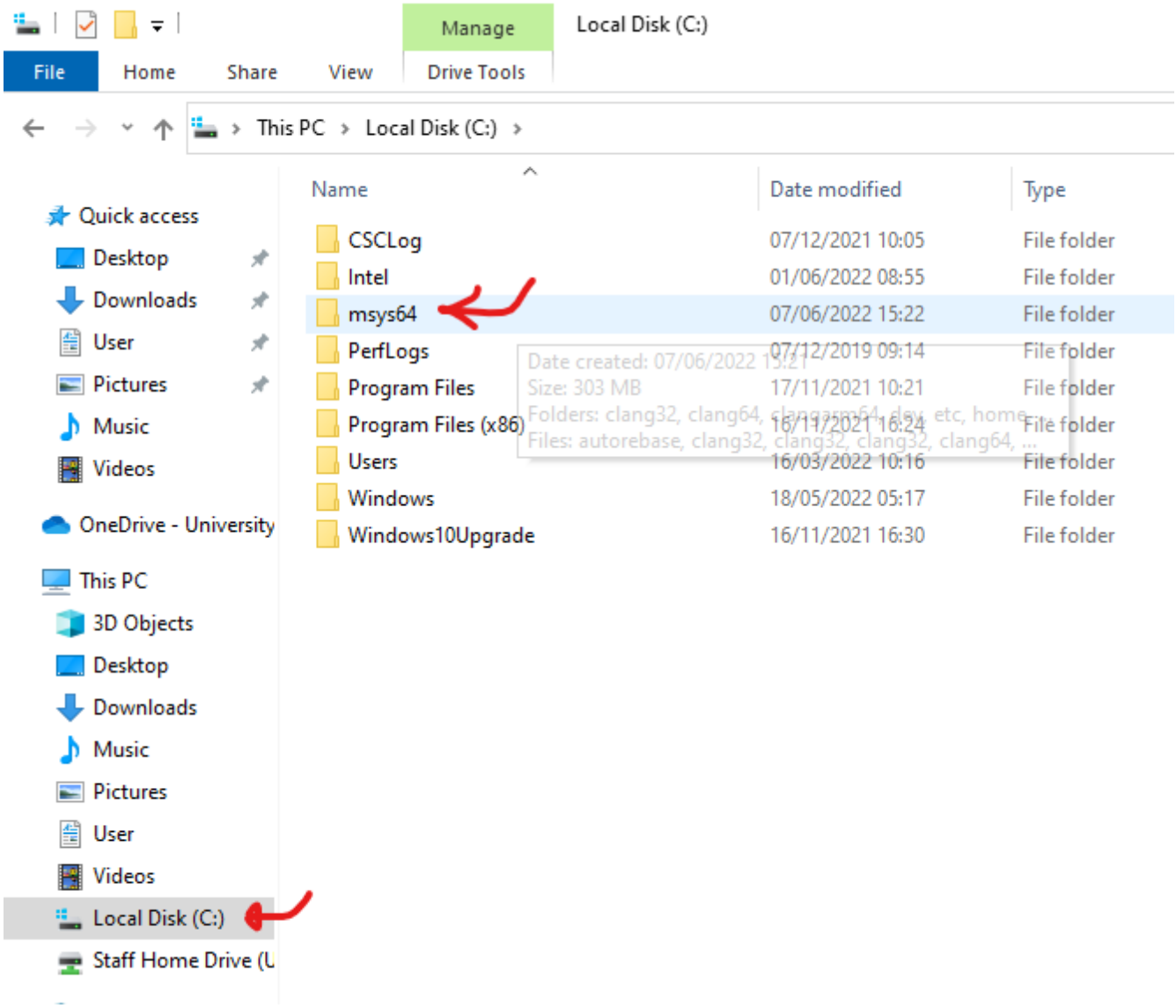

Figure 2.2: Accessing the contents of the ‘msys64‘ directory via Windows File Explorer. Later on, the simulation code and simulation output will be found within this directory (in particular, within its ‘home‘ directory).

2.8 Mac: a UNIX terminal is provided via the Terminal app within the Applications/Utilities folder

To use the UNIX toolchain on a mac, you will need to use a UNIX terminal. Luckily this is installed as per default on any mac distribution. You can find the Terminal app in the `Applications` > `Utilities` folder on your hard drive. See Apple's support page here for more information on how to open this application.

2.8.1 Mac: Install `g++` and `git`

In order to install g++ on a Mac, you need to follow quite some instructions. Once this is installed `g++` is installed, as is `git`.

2.8.2 Mac: install a text editor like textmate

You can find Download textmate here. If you don't like textmate, find any other code editor that supports syntax coloring of C++,R,Python files and supports `CR/LF` and `LF` line endings.

Chapter 3

A typical work cycle when running simulations in C++

Here we illustrate some of the basic steps we take to run simulations and analyze them. A typical work cycle consists of

i) accessing the code base via a UNIX terminal,
ii) making sure you have all the latest updates of your code base using `git pull`,
iii) editing the code via an appropriate text editor
iv) compiling the code
v) running the compiled program
vi) analyzing the resulting data

3.1 Access the code base via the terminal

3.1.1 Starting the UNIX terminal

A unix terminal is a command-line environment that allows you to interact with your operating system and the files on your computer. By now, you should be able to start your **UNIX terminal programme**, which should be one of the following:

3.1.2 Your terminal program on windows

- MSYS2-64 on your local windows PC with which you can access the local code repository. See **??** for install instructions
- mobaxterm on your local windows PC to log into a remote server. This should be either somewhere in your `Downloads` or `Desktop` folders, dependent on where you downloaded it. See 2.5 for install instructions.

3.1.3 Your terminal program on a mac

- The Terminal programme that can be found in the `Applications > Utilities` folder

If you can't find your UNIX terminal, read section **??** and if that still does not work, get in touch.

3.1.4 Use the terminal to navigate to your repository

First, we need to go to your home directory. The home directory is contained in the alias `~` and if you are not already there, it can be accessed by using the `cd` command (which stands for 'change directory') in the following way:

```
cd ~
```

You can then use the `pwd` command (where `pwd` is an abbreviation of present working directory) to inspect the current directory at which you are. If everything is ok, you should see that you are in your home directory:

```
pwd
```

** Note for windows users: ** the MSYS2 home directory is not the same as your windows home directory. See for more information 2.7.2.

3.1.4.1 Check the contents of your home directory using `ls`

To navigate any further (remember: we need to go to your code repository), we need to get an idea of where to go in the home directory. To this end, we can check the contents of your home directory using the `ls` (list files) command. Typially, I provide the `ls` command with some additional 'flags' to increase the amount of information provided by `ls`. One can provide such flags by adding a space followed by a dash `-` and then followed by some additional single-character modifiers that change the behaviour of the `ls` command. For example, we can type

```
ls -alnh
```

Here we used the flags `-alnh` to make sure (i) we list hidden files (`-a`), (ii) we list all files below each other (the long format: `-l`) rather than dumping all files together on a single line, (iii) we list all files with numeric user and group IDs (`-n`) and finally (iv) we list all dates and numbers in a human-readable format (`-h`).

If you want to know more about the documentation of the `ls` command, type:

```
man ls
```

which provides you with a manual page (man page) of the `ls` command. You can close this man page and return to the command line by typing `q`.

3.1.5 Using `ls` to locate the github repository that contains your simulation code

All research projects in this lab use code repositories that I have uploaded to github and that you can download to your own computer and modify. See here for a list of all repositories. If you don't know which of these repositories to download to your own computer, get in touch.

Working with code repositories on github has several advantages: for starters, it facilitates carrying through updates without having to e-mail back-and-forth umpteen different versions of the source code. Second, if you have your own github account and I have provided you with write access to my code repository, you can then submit your own updates to my github repository, so that we all have the latest version of the code available. Third, github is great when it comes to keep track of all changes, which is essential when you want to roll back changes, for example. Fourth, it makes sure that the science we are doing is accessible and hence hopefully a bit more transparent.

If you vaguely remember using the `git clone` command before, perhaps you should first inspect the output of the `ls -lanh` command from within your home directory (`cd ~`) to find out whether a copy of the github repository in question is already present within your home directory. For example, the command below shows code that displays the contents of a home directory on a remote computer:

where you see several github repositories listed. If you want to understand what all the columns are, have a look here. Note that the `d` modifiers in front mean that the file is a directory (as is the case for the `coop_size` directory, for example), whereas a `-` in front means the listed file is a regular file (as is the case for the `summary_data.csv` file). Git repositories are typically contained in directories unless we have messed things up.

If you indeed find a directory within your home directory that looks remarkably similar to one of the repositories, it means you have downloaded the repository already before. Hence, you can skip section 3.1.6.

3.1.6 Download the simulation code by using `git clone`

To download a copy of the remote repository containing the simulation code to your own computer, execute the following command:

```
git clone https://github.com/bramkuijper/YOUR_REPOSITORY
```

where `YOUR_REPOSITORY` is the name of the repository that you should be using from this list. By now, you should know the name of the repository you should be cloning, otherwise get in touch.

3.2 Exploring your local copy of the repository

Once everything is cloned to your local hard disk, you can explore its contents. Use `cd $repository_name` (where you need to replace `$repository_name` with the name of your repository, for example `cd coop_size`) to enter into the repository.

3.2.1 Pro-tip: save on typing a lot by using `Tab` autocompletion

Typing out whole directory names like `cd sexsel_space_multisignal` gets tedious after a while. Luckily, you can save on typing everything. Instead type the first few letters like `cd s` and then pressing the `Tab` button to autocomplete the directory name.

If there is only one directory name that starts with an `s`, pressing `Tab` results in immediate autocompletion of the directory name. If however, there are multiple directories starting with an `s`, the terminal will output all the different directories starting with an `s` (after pressing `Tab` at least twice). For example, typing `cd s` in the directory that has contents as listed in the above listing **??** and then pressing `Tab` twice will give you

```
cd s
#sexsel_bacteria/              sexsel_space_multisignal/     sex_specific_maternal_eff
```

what this means is that the terminal found four alternative directory names that start with an `s`. It then asks you to supply more characters so that it knows which directory name to autocomplete. Hence if I'd type `cd sexsel_s` followed by `Tab`, there are no alternatives left, and the terminal immediately autocompletes the remainder of the directory name `sexsel_space_multisignal`.

3.2.2 Exploring your local repository continued...

If the `cd` command was successful, you can then use `ls` to explore the contents. With respect to the `coop_size` directory, for example, we have the following listing:

```
ls -alnh
total 564K
#drwxr-xr-x 1 848446593 848298497  336 May 26 10:13 .
#drwxr-xr-x 1 848446593 848298497  178 May 24 00:48 ..
#-rw-r--r-- 1 848446593 848298497 4.8K May 24 00:48 coop_fluct.lyx
#-rw-r--r-- 1 848446593 848298497 9.6K May 24 00:48 coop_size_continuous.nb
#-rw-r--r-- 1 848446593 848298497 497K May 24 00:48 coop_size.nb
#-rw-r--r-- 1 848446593 848298497  16K May 24 00:48 coop_variable_envts_leggett.nb
#-rw-r--r-- 1 848446593 848298497 3.6K May 24 00:48 coop_variable_envts_queuing.nb
#drwxr-xr-x 1 848446593 848298497   32 May 24 00:53 data
#drwxr-xr-x 1 848446593 848298497   94 May 27 10:24 figs
```

```
#drwxr-xr-x 1 848446593 848298497  204 May 27 00:05 .git
#-rw-r--r-- 1 848446593 848298497  123 May 24 00:53 .gitignore
#drwxr-xr-x 1 848446593 848298497  126 May 26 14:12 sbin
#drwxr-xr-x 1 848446593 848298497   98 May 24 00:53 src
#-rw-r--r-- 1 848446593 848298497  19K May 24 00:53 varying_patch_size_overlap.lyx
```

There is a whole bunch of files here, but we typically want to be looking into the `src` directory, which contains - in turn - other subdirectories such as `ibm` or `numerical` with source code. By now, you should

Alternatively, you can also your file explorer programme (Windows File Explorer or Finder) to go the location of your repository (see section 2.7.2).

The files that end in `.cpp` are the C++ source files, which contain the necessary code. The files that end in `.hpp` are C++ header files, which provide a blueprint to the C++ compiler about the interface of the different classes and structs used. Next, you may also see a file such as `CMakelists.txt` or `makefile` which are used by `cmake` and `make` to handily automate compilation of all the files.

3.2.3 Inspecting the source files

You can inspect the contents of the source files by opening them in Notepad++ (on windows, see section **??**), which you can open through the windows start menu. Within notepad++ you then have to navigate to the location of your MSYS2 home directory (see section 2.7.2.

On a mac, you can inspect the contents of the source files by opening them in textmate, which you can find in your Applications folder.

3.3 Updating the software repository

If you would like to update your repository to contain the latest version, `cd` to the directory of the repository on your local computer and obtain the most recent version of the repository. You do so by typing

```
git pull
```

If you see the following message:

```
Already up-to-date.
```

you already have the latest version of the source code in your repository.

3.3.1 Conflicts when updating the software repository

It may be that you get messages like

```
git pull
#error: Pulling is not possible because you have unmerged files.
```

```
#hint: Fix them up in the work tree, and then use 'git add/rm <file>'
#hint: as appropriate to mark resolution and make a commit.
#fatal: Exiting because of an unresolved conflict.
```

in which case you might have files in which you have made local edits which clash with any remote edits. This is called a 'conflict', as apparently, your edits conflict with remote edits. There are two options:

1. If you deem your edits to not be so important: Use `git` to overwrite the edits with the version from the remote repository. To this end, `cd` into the folder of your repository and do the following:

```
git checkout --theirs .
```

which tells `git` to checkout files from the remote repository (`--theirs`) and replace files in the local repository (`.`).

2. If your edits are very important If you open your files in your editor, you should find lines that highlight

```
>>>>
some text
====
some text
<<<
```

which reflect the various edits. Try to make choices about what you find important and make sure that the `>>>>`, `===` and `<<<<` characters are removed from the file.

3.4 Compiling the software

For a typical simulation, you will find the source code in the `src` directory (or one of its subdirectories) within the repository. If you want to compile the simulation code,

Chapter 4

Working with `git`

Here a listing of how to work with git and some common errors:

4.1 Recording local changes to remote via `Git commit`

For `git` to be aware that you are making any changes, you need to record them by typing the command

4.2 Uploading local changes to remote via `Git push`

If you want to upload your local changes to the remote repository, we first need to make sure that we have done a `git commit` so that all changes are recorded in the repository. Subsequently, we do

```
git push
```

we try to upload our local changes to the remote repository.

4.2.1 `Issues withgit push: 1. remote containswork that you do not have locally After runninggit push, it might be that the remote repository contains more recent changes than are included in the current code.git‘` then throws the following error:

```
To github.com:bramkuijper/sexsel_space_multisignal.git
```

```
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'github.com:bramkuijper/sexsel_space_multisignal.gi
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

To solve this, we need to

Chapter 5

Some Writing Advice

Writing your dissertation or literature review can be a challenge. Here some advice:

5.1 Read about writing: literature that helps improving writing skills

We all read books to improve our writing, even when you are a native speaker and think you have seen it all! (hint: you haven't.) Reading about writing techniques is a great way to improve your skills.

If you want to know more about scientific writing, perhaps the short book by Mack (2018) might well be worth trying. This book is freely available online here. Another good book on how to write scientific papers is Gastel and Day (2022).

For a literature review, it can be helpful to read the paper by Sayer (2018), which is all about how to write review papers.

Next to learning more about the techniques of scientific writing, it might be helpful to improve your general writing skills too. Several good books are available. A first one is Williams (1990) which has later on appeared with different titles. Other books are Pinker (2014) and Zinsser (2006). A classic is Strunk (1959).

Hard copies of most of these books are available at the library or in my office. Of course, you should not be looking around the internet for pdfs of these books!

5.2 Example theory papers to help you write about theory

Papers on theory are a bit different than empirical studies. Hence, here a bunch of examples to see how theory papers are typically written: (Fawcett et al., 2007),(Trimmer et al., 2015),(Kahn et al., 2015).

In a theory paper it is important to explain the parameters that you use and why. In part, such explanations may focus on a comparison to previous theory, as in: "To compare our model to predictions made by the classical hawk-dove game (Maynard Smith and Price, 1973), the cost of losing a fight c and the value v of winning a fight are modeled as unidimensional variables with $c > v$.". Alternatively, you could refer to the empirical literature, as in: "In speckled wood butterflies, it is unlikely that $v > c$, because territories are far too temporary to be of any value."

5.3 Common examples where writing goes wrong

5.3.1 Back up qualitative statements

Avoid sentences like "The random matrix method introduced by May 1977 has been highly important in theoretical ecology. Here we use this technique to understand how mutualisms affect ecosystem dynamics". The first sentence makes a claim about importance, but does not back it up with a statement that indeed testifies of its importance (one may add: "as this approach has been central to the analysis of ecosystem stability in several later studies [citations]"). But even then, you could argue that it still tells very little: why not tell us instead what the random matrix method is about and what it does?

5.3.2 Being overly verbose

Avoid sentences such as "Territory productivity can be measured by a variety of indirect and direct methods (Davies et al 2012). These measures can be used to calibrate simulation models." The mentioning of "indirect and direct methods" adds very little here. Why not give at least an example of an indirect and a direct method? Or perhaps scrap the indirect vs direct and just focus on 'different methods' and then give an example of such a method. Or if you realize the sentence on different methods adds too little, why not omit it altogether?

5.3.3 Make sure each paragraph addresses the broader question

It is easy to get mired into examples or definition questions that are a bit particularly. Unless you make explicit why the paragraph contributes to insight

about the broader question asked in your dissertation, consider what you are writing irrelevant.

5.3.4 Clearly label figures

If you use a figure with different panels, each panel should have a label, such as "A", "B", "C", etc. There are no excuses. If your R package does not do that, edit your figure in a graphics programme, or look at some of the queries about labeling figure panels in `ggplot`, for example here.

5.3.5 Reference all figures

If you don't refer to a figure in the text, you don't need it. Throw it out.

5.3.6 Explain terminology upon first use

Typically, assume that the audience is someone who knows quite something but is not necessarily a scientist. For example, a 2nd-year undergraduate. So any terminology needs to be explained, let alone any abbreviations upon first mentioning.

5.3.7 Various writing errors:

- It is evolutionar*ily* stable strategy, not evolutionary stable strategy
- Temperature-related perturbations instead of temperature related perturbations

Bibliography

Fawcett, T. W., Kuijper, B., Pen, I., and Weissing, F. J. (2007). Should attractive males have more sons? *Behav. Ecol.*, 18(1):71–80.

Gastel, B. and Day, R. A. (2022). *How to write and publish a scientific paper*. Greenwood, Santa Barbara, CA.

Kahn, A. T., Jennions, M. D., and Kokko, H. (2015). Sex allocation, juvenile mortality and the costs imposed by offspring on parents and siblings. *J. Evol. Biol.*, 28(2):428–437.

Mack, C. A. (2018). *How to Write a Good Scientific Paper*. SPIE.

Maynard Smith, J. and Price, G. R. (1973). The logic of animal conflict. *Nature*, 246(5427):15–18.

Pinker, S. (2014). *The sense of style: the thinking person's guide to writing in the 21st century*. Viking, New York, New York.

Sayer, E. J. (2018). The anatomy of an excellent review paper. *Functional Ecology*, 32(10):2278–2281.

Strunk, William, J. (1959). *The Elements of Style (4th edition)*. Allyn & Bacon, Boston, MA.

Trimmer, P. C., Higginson, A. D., Fawcett, T. W., McNamara, J. M., and Houston, A. I. (2015). Adaptive learning can result in a failure to profit from good conditions: implications for understanding depression. *Evolution, Medicine, and Public Health*, 2015(1):123–135.

Williams, J. M. (1990). *Style: toward clarity and grace*. University of Chicago Press, Chicago.

Zinsser, W. (2006). *On Writing Well: The Classic Guide to Writing Nonfiction*. HarperCollins, New York.