4. Local Software Installation (Optional)

Z620: Quantitative Biodiversity, Indiana University

OVERVIEW

Quantitative Biodiversity (QB) will rely on Posit.Cloud as an integrated environment for using R, RStudio, Git, and LaTeX. While this is convenient – especially for learning in the classroom – you may want to install these programs on your personal computer. The following guide provides instruction on how do this.

1) R

R is used to manipulate, visualize, and statistically analyze data. Install the latest version of R (January 2025: v 4.4.2; "Pile of Leaves") using the following instructions:

1. Open a web browser and navigate to the IU CRAN mirror: http://ftp.ussg.iu.edu/CRAN/
2. Select your operating system (Mac, Windows, Linux) For Windows Users: install the `base package` For Mac Users: install the package for your current version of OS X. You can find out what version is installed by going to the Apple menu and choosing "About This Mac". The version number of OS X you are using appears directly below the words "OS X."
3. The default installation options are recommended for most users.

Mac users please read: The most recent versions of R no longer come packaged with an X11 graphics device. Instead, it relies on the program XQuartz for generating plots, etc. If you do not already have XQuartz installed, you will need to do this. You can download XQuartz v2.8.4 from the following site: http://xquartz.macosforge.org/landing/

2) RStudio

RStudio is the most popular development environment to conduct exercises in R. It also can edit and create Markdown files. To install RStudio: please do the following:

1. Open a web browser and navigate to https://posit.co/download/rstudio-desktop/.
2. Select and download the appropriate installer for your operating system (Windows, Mac, Linux).
3. Open the installer and follow the on-screen directions.
4. The default installation options are recommended for most users.

3) LaTex

LaTeX is a useful typesetting system. LaTeX can be used via RStudio and the R package **`Knitr`**, which converts our RMarkdown files (.Rmd) into professional-quality PDF files. This will happen each time we use the "Knit" button in RStudio. This means that we need to have LaTeX installed, along with a few other packages.

Note that the `Knitr` conversion will output a .PDF file with the same name as the processed .Rmd file. This cannot be completed if the target file is being used by another program such as a PDF reader.

If you do not have LaTeX installed on your computer, you will need to do one of the following:

Install TinyTeX
This is a cross platform LaTeX distribution developed with R users in mind (https://yihui.name/tinytex/). The `tinytex` R package can be used to install TinyTeX from the R console quite easily with the following two commands:

```
#requires web access
install.packages('tinytex')
tinytex::install_tinytex()
```

During installation a couple of error messages may pop up. These may be ignored, just hit the `OK` button and the installation should continue. It takes several minutes to complete. Once the installation is complete you should restart RStudio and check if TinyTex properly installed by typing `tinytex:::is_tinytex()` in the R console. If the returned value is `TRUE` you should be good to knit .Rmd files to PDF.

Alternative installation method for TinyTex

If for some reason TinyTex fails to install on you machine you can try the following:
On a Mac:

1. Install Basic Tex: https://tug.org/mactex/morepackages.html
2. Note: this will require you to run the commands below (to install `framed` and `titling`).

On a PC:

1. Install Basic MiKTeX: http://miktex.org/download.
2. Note: you need to use the MiKTeX package manager to download required style guides.
3. Add folder containing MiKTeX binaries to your **PATH environment variable**

By default, RStudio uses style guides to format our PDF documents. These style guides include framed.sty and titling.sty. We have found that not all LaTeX installations include these style guides. If you do not have them, you will get an error message when you "Knit". To fix this, you need to install the required files. For instructions to add MiKTeX binaries to **PATH environment variable** see https://stackoverflow.com/questions/47911135/r-sweave-no-tex-installation-detected.

On a Mac: Type the following in terminal

```
tlmgr init-usertree
tlmgr --usermode install framed
tlmgr --usermode install titling
```

The `tlmgr` command should work from any directory if the installation was successful.

If for some reason this does not work (likely due to a permission setting in your operating system or the command PATH wasn not properly loaded during installation), try changing to the MacTex directory (found in the MacTex readme file) to enable `framed` and `titling`. Most likely adding "./", which will indicate to Unix that the command is present in the directory, will allow the `tlmgr` command to function.

```
cd /usr/local/texlive/2022/bin/universal-darwin
./tlmgr init-usertree
./tlmgr --usermode install framed
./tlmgr --usermode install titling
```

Adding "sudo", which allows the user to run a command with elevated privileges, may be required to run the `tlmgr` command, depending on your computers administrative permission settings.

```
sudo ./tlmgr
```

On a PC: You have two options:

1. Open the MikTeX package manager from Start.
2. Search for and install the following: framed, titling

   OR

3. Type the following in command line (or GitBash):

```
mpm --install framed
mpm --install titling
```

4) Git

Downloading Git

To use Git with R/Rstudio, ensure you are using the most up-to-date version. If you do not have Git yet installed, please do the following:

1. Open a web browser and navigate to git-scm.com/download/
2. Select the appropriate operating system.
3. The download should start automatically.
4. Open the installer and follow the on-screen directions.

On Mac: You will need to make sure you have Xcode Command Line Tools installed. To test this type `which g++` at the command line (i.e., Terminal). If you get `/usr/bin/g++` as a reply, then you are ready to move on.

On Windows: This process will install Git Bash (msysGit). During installation, you will be asked to adjust your **PATH environment variable**. To provide you with the most flexibility, we recommend that you select the option to "Use Git from the Windows Command Prompt". In addition, we recommend that during installation you select "Use OpenSSH" for your secure shell client with GitBash.

```
During installation, you will be asked how to configure the line-ending conversions
```

On Mac: We recommend "Checkout as-is, commit Unix-style line endings" **On Windows**: We recommend "Checkout Windows-style, commit Unix-style line endings"

Conifguring Git

1. **Organization:** We recommend that you create a folder in your user directory called '*GitHub*' to make this and future assignments easier to manage. In the following code, you will move to your user directory using the `cd` (change directory) function. The *slashdot* (./) specifies your current directory (../ would specify the directory above your current directory) and you will make a new directory with `mkdir` function.

```
cd ~
mkdir ./GitHub
cd ./GitHub
pwd
```

2. **User Configuration:** You will need to configure your local Git installation. You will do this by entering your name and email. You will also set two parameters: push.default and credential.helper

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
git config --global push.default simple
git config --global credential.helper store
git config --global core.editor "nano"
```

3. As described above, during installation, you will be asked about how to configure line endings in Git. Line endings are invisible characters that your operating system places at the end of each line in a document. On Unix machines (e.g., Mac), this is the linefeed character (LF).

On Windows machines, this is the carriage-return (CR) and linefeed (LF) characters. This difference in line endings between Mac and Windows causes incompatibilities between the two systems.

However, Git is equipped to handle the differences by silently converting line endings when repositories are pushed to remote servers. We *strongly* urge that you configure this behavior in order to prevent any future issues (i.e., conflicts) when collaborating across computer platforms.

```
**On Mac**

‘‘‘sh
git config --global core.autocrlf input
‘‘‘

**On Windows**

‘‘‘sh
git config --global core.autocrlf true
‘‘‘
```

Using SSH with Git

While QB adheres to a philosophy of openness and collaboration, it is also important for institutions and individuals to protect their data and computing environments. One of many ways to achieve this type of security is to use **SSH keys**. SSH stands for *secure shell*. To use SSH, you generate a pair of keys: one public and one private.

Note: if you want to connect to GitHub with SSH keys, you will need to complete the following steps for *each* computer that you use.

1) Generate an SSH Key

Using Rstudio One way to generate an SSH key is through RStudio. Go to Preferences and choose the Git/SVN tab (with box-looking icon). Depending on the version of RStudio on your local computer, this same tab is found in a dialog box that is accessed by opening the Tools tab and clicking on Global Options. Make sure the box is checked for `Enable version control interface for RStudio projects`. In the Git executable box, it may say `/usr/bin/git` and SVN executable box may say `usr/bin/svn`. For Windows users the Git executable box may contain `C:/Program Files/Git/bin/git.exe`. In the `SSH RSA Key` box, you should type `~/.ssh/id_rsa/`. You can then hit the button that says `Create RSA Key...` You can now view the public key. Copy this, so you can paste it into GitHub.

Using Terminal Alternatively, you can generate an SSH key by typing the following at the command prompt:

```
ssh-keygen -t rsa -C "your_email@example.com"
```

You will be asked to enter and re-enter a passphrase. After that, you need to add the new key to the SSH-agent using the following commands, which will generate an agent pid (process identifier).

```
eval "$(ssh-agent -s)"

ssh-add ~/.ssh/id_rsa
```

To obtain the SSH key you just generated, type the following command at the Terminal. (Note: your key may be named one of the following instead of id_rsa.pub: id_dsa.pub, id_ecdsa.pub or id_ed25519.pub)

```
pbcopy < ~/.ssh/id_rsa.pub
```

2) Put SSH Key Into Github

Log in to your GitHub site at www.github.com. In the upper right hand corner, click on your profile and choose `Settings`. Now, on the left hand side under `Personal Settings`, click on `SSH and GPG keys`. (Alternatively, type https://github.com/settings/keys) In the upper right, click on the green button that says `New SSH Key`. On the new page that opens, add a descriptive title (e.g., Jay's MacBook). Now paste the SSH key into the key window and hit the green `Add SSH Key` button. You may be asked to supply your GitHub password.

3) Add Key to Local Computer Key Chain

On Mac: Open the terminal and type the following. This will add your ssh key to Apple's keychain (K)

```
ssh-add -K ~/.ssh/id_rsa
ssh-add
~/.ssh/id_rsa
```

On PC:

For windows users who are using git bash, the above "ssh" commands will all end in .exe (e.g. `ssh-keygen.exe`).

Additionally, `pbcopy` is a Mac specific program, but you can use `clip.exe` command instead:

```
clip.exe < ~/.ssh/id_rsa.pub
```