CC Primer: Compute Canada for geoscientists in a rush

Ricardo Barros Lourenco

Last revision: 2022-02-23

Contents

Chapter 1

Front Matter

1.1 Copyright

1.1.1 How to cite this work

Please use this BibTeX fragment:

```
@book{BarrosLourenco2022,
  title     = "CC Primer: Compute Canada for geoscientists in a rush",
  author    = "Barros Lourenco, Ricardo",
  year      = 2022,
  doi       = "10.5281/zenodo.5937906"
  note      = {\url{https://ricardobarroslourenco.github.io/CCprimer/}(visited yyyy-mm-dd)}
}
```

Obs.: Note that the DOI is of a previous version (10.5281/zenodo.5937906) when comapred to the badge displayed on this website, because it relates to all versions of this document, rather than a especific one.

1.2 Cronology

2022-01-31 - Beginning of v0.1. (This version is quite unstable, and versioning during this stage will be limited).

Chapter 2

Version Control Systems

Version Control Systems Wikipedia contributors [2002] (VCS - also named as *Revision Control*, *Source Code Control*, *Source Code Management*) are Computer Systems traditionally used to manage the complexities of software development, majorly Computer Systems involving multiple software modules, and large development teams.

Such systems help to organize such development, especially when collaboration of multiple developers is done, and several times people are doing code changes in a same, or close, part of the code, which may break it, or even generate unexpected results that would not be traceable.

2.1 VCS: Local repository vs. Remote repository

Common VCS's (ex.: CVS, SVN and Git) always are client-server systems.

The VCS client is not merely a means to access a central repository hosted at the VCS server, but actually is part of running checks and balances that are necessary when adding code to the central repository. A situation that demands that is when multiple users are doing a change in a same piece of code, and checking it only at the central repository may complicate more a situation already complicated (simultaneous contribution). Therefore, it hosts, locally a local copy of the codebase, which reflects a view of that codebase in time (more preciselly when that local user retrieved a copy from the central repository for editing).

Once the changes are done, the user saves a local copy of these at the local repository, and if these meet the criteria for changes (and this may vary a lot among systems), the user is able to persist this change at the local repository,

as a new version of the codebase. This process is known as *commit* (in this case, a local one).

2.2 Git

Git is a contemporary VCS that is used as the backbone of operations run on GitHub. So, when planning to operate with GitHub, it is almost certain that you will need to install a Git client on your machine.

We can say *almost* because GitHub provides an own client, GitHub Desktop which provides a Graphical User Interface(GUI) to a git client.

We will not cover such GUI usage since on Compute Canada you will not have access to GitHub Desktop on their machines, but only to a regular Git client (that you can use to access either GitHub, or another Git compliant server such as a private repository built with GitLab).

2.2.1 Installing a git client on your machine

2.2.1.1 Ubuntu (or any other Debian-based environment)

First update your Operating System (OS) repository indexes:

```
$ sudo apt-get update
```

Then proceed to install:

```
$ sudo apt-get install git
```

Note: If you use Ubuntu, git is already provided as a base repository on this distribution. If you use another linux, and this does not work for you, please open an issue on this project and use the label *requests*, and I will try to solve and include here for future reference.

2.2.1.2 Apple

Apple is an OS that not use repositories by default, and git is not provided in the main set of software. So you would have three main options to install git:

- Install Xcode (*Preferred*): Apple has a software development kit (SDK) named XCode which provides git among several other software for MacOS and iOS software development. The advantage of using XCode´s git distribution is that it comes adjusted for your OS and is supported by Apple, whcih avoids conflicts and security issues.

  - To install XCode (and git by consequence), open the App Store, and search XCode as a software developed by Apple, and install it. To test, open the Terminal App, and run *git*. You should receive an output of basic description of commands available on git.

- Install Homebrew, and then install Git (if you are a Homebrew user, or uses a lot of *.nix software not supported by Apple it is useful):
  - Download and install homebrew from their website;
  - Install git as:

```
$ brew install git
```

  - In the terminal, run *git* and you should receive an output of basic description of commands available on git.

- Install a standalone version of git (*not recommended*): You can download and install from git´s website a standalone client (I will not be covering this approach, because git will not be updated via this kind a install, posing a security risk and install this at your own risk - and effort!)

2.2.1.3 Windows

Since Windows works with standalone installations, the install follows as this:

- Go to the Git website and download the latest client (download the standalone version, unless you have severe storage limitations on your machine);
- Once downloaded, run the installer with admin permissions and follow the default installation;
- When installed, open PowerShell (PS) (or the command prompt, if you do not have PS installed), and run *git* and hit enter. You should receive an output of basic description of commands available on git.

2.2.2 Setting up your local git client

Once you have your git client up and running, you need to setup the access credentials to connect to a git repository, such as GitHub or GitLab for example.

To do so, run on your bash terminal (or command-line if on windows):

```
$ git config -- global user.name "Your full name"
```

Note: On quotes you need to specify a name (depending on the environment, it should be your full name, or an alias, such as employee number, for example).

This command has a global scope, so all users on your machine would have the same setup.

Then you need to specify an e-mail address (if on GitHub, it is preferred that is linked to your account):

```
$ git config -- global user.email "Your e-mail address"
```

Then check, if all values were set:

```
$ git config --list
```

You should see an output that reflects those previously set name and e-mail values.

2.3 GitHub

GitHub is a Web collaborative Version Control service based on a Git platform. It was created in 2008, and as of today is the largest repository of source code in the World. Aside of VCS capabilities, it provides other computational services, such as Continuous Integration (CI) and Continuous Deployment (CD), Web Page Hosting on GitHub Pages, security solutions, a software marketplace, among many others. It was recently acquired by Microsoft, as one of the largest transactions in the tech domain.

GitHub has several different tiers of access, from basic free accounts up to corporate ones.

A main advantage for academic users would be using the GitHub Education which includes a free GitHub PRO account and several software for free (or services credit hours) while you are enrolled in an academic institution. GitHub Education also have different tiers of users aiming students, teachers, and the academic institutions itselves. Once you create a simple free GitHub account, you can return back to GitHub Education, and request to be enrolled in the program.

2.3.1 Creating an account

To create an account is simple:

- Go to GitHub pricing page, select the Free tier, and click *Join for Free*;
- Just follow the prompts to create your personal account.

- You should receive an e-mail from them on the registered e-mail account, to verify your identity. If you fail to do so, your account would be basically useless.

Note: Since I have created my account some years ago, I am just using GitHub's user documentation as reference. If things get complicated in this step, please let me know, and I will expand here.

2.3.2 Insert GitHub credentials on your local Git

Once you have your GitHub account created, and verified, it is time to setup these credentials on your local Git client. While logged on your GitHub account you should:

- Click on your *name/avatar/photo at the upper right corner of the screen*, and then click on *Settings*;
- Then click on *Developer Settings* (it is the last item on the bottom of items at the left panel);
- Now click on the bottom item, *Personal access tokens*;
- Then, click on *Generate new token*, you will be requested again for your password on this step;
- Now you will be required to fill in some info:
  - *Note*: This will be a name of this Token. I often create a Token per software+device I use, then someone can write "Rstudio on Laptop", to differentiate from "Rstudio on laboratory desktop".
    - Isolating tokens on different (software,device) pairs helps isolating accesses to your GitHub account, and is important to contain a security breach. If someone is able to fetch one of your tokens, you will know from which machine and which software on it came from.
  - *Expiration*: You should define a expiration date for your token. Choosing a date is an open ended question, but ideally systems exposed to the Web, or multiple users such an HPC cluster, should be changed frequently.
    - *Avoid at all means to use the No expiration* mode, because you never want to forget unattended keys of your GitHub (ex. You graduate, and your key is left on a machine that another student will use in the lab).
  - *Select scopes*: Perhaps this is the trickiest setup. The definition of a Token is actually a definition of a OAuth Token. On GitHub, this implies on being able to select all options that a user has in terms of operations on the platform, and actually, narrowing down to what a user wants to grant permission for in such Token.

    - Note: It is tempting to grant *all permissions* to a single Token, but the user should ask it that is really necessary. In one end, granting all permissions is too permissive, and being as problematic as having a token with no expiration data.
    - To look into what each scope covers, please look into this page.

- Once you have generated your token, treat is as a password (even in terms of security/sharing/etc.). You should not persist a copy of it, but only use for your local git setup. Even if you loose access to it, you can always revoke that token, and create a new one.
- To set your local git to access GitHub (remind that you need first to set global variables), you just need to access GitHub with it. A simple way, that will be further explained in more detail would be making a local copy of a repository using *git clone*:

```
$ git clone https://github.com/a_very_weird_user_name/a_more_weird_user_project_name.g
```

A concrete example would be the source-code of this document:

```
$ git clone https://github.com/ricardobarroslourenco/CCprimer.git
```

- Once done, your local git should request:
    - User-name: The one you have set for your GitHub account;
    - Password: Your token.
- After this, it should make a local copy of that cloned repository (more especifically this copy will be stored at the directory path you have run the clone command - in bash / terminal / command line).
    - Note: If you have not been requested a username/password, maybe you have already a GitHub account already set on your machine (so please note if any errors occur when trying to access GitHub with git, no username + password is required - if these occur, they will be output on your bash / terminal / command line)
- Finally, to persist the token on your local install, run:

```
$ git config --global credential.helper cache
```

- If you need to clean-up the token(s) installed:

```
$ git config --global --unset credential.helper
```

2.4 A combined usage of Git-GitHub

This section describes a usage of Git-GitHub intended for scientific usage. Therefore, is important to remark that there is still no consolidated practice on such application, considering that these tools were not meant for academic usage, but rather for a software engineering one. Such application varies across research groups and scientific domains, and on a best effort, we will summarize some good practices and update as needed.

Chapter 3

Compute Canada

Note: This section is under heavy work. For now, please refer to Compute Canada's Wiki.

3.0.1 Running batch jobs

If you want to run a R-language batch job, please take a look on the CCrecipes repository.

More specifically at the `slurm/R/r_batch_standalone.sh` file, we have:

```
#!/bin/bash

### Sets shell for parallel program (no distribution - no MPI)
### Inspired by the current documentation available on CC Wiki.

#SBATCH --account=def-someacct             # replace this with your PI account
#SBATCH --nodes=1                          # number of node MUST be 1
#SBATCH --cpus-per-task=4                  # number of processes
#SBATCH --mem-per-cpu=2048M                # memory; default unit is megabytes
#SBATCH --time=0-00:15                     # time (DD-HH:MM)
#SBATCH --mail-user=yourname@someplace.com # Send email updates to you or someone else
#SBATCH --mail-type=ALL                    # send an email in all cases (job started, job ended,

### Load library modules
module load gcc/9.3.0 r/4.0.2

### Load r-packages (builds locally) - comment, if unecessary to install new libraries
#R install_script.R
```

```
### Export locally installed packages
# TODO: check if this path holds
export R_LIBS=~/local/R_libs/

### Run main_job.R - or any batch script necessary
R CMD BATCH --no-save --no-restore ~/main_job.R
```

Some requirements:

- The R file `main_job.R` needs to be at the root of your user folder.
- Also the script needs to be executable (command runned at the same folder as `r_batch_standalone.sh`:

```
$ chmod +x r_batch_standalone.sh
```

- Finally, you can submit this batch job as:

```
$ sbatch r_batch_standalone.sh
```

Once submitted, you should receive a job number at the console. Once its status change at the scheduler (job started, job ended, job aborted), you should receive an e-mail.

If getting into running issues, take a look on the errors by inspecting the output files which will be saved at the same folder you submitted the job and have the name `slurm-xxxxxx.out` in which `xxxxxx` is the job number.

Bibliography

Wikipedia contributors. Version control — Wikipedia, the free encyclopedia, 2002. URL https://en.wikipedia.org/wiki/Version_control. [Online; accessed 01-February-2022].