

# Working with collaborators (including through Overleaf)

August 6, 2022

You may work with a group of co-authors and share your project's folder using GitHub, Dropbox, or a similar file sharing system. It is possible that the work is divided in such a way that some collaborators work only on the text, and are not involved in, or just cannot run the code. For example, they may not have Python, R, Julia, or Matlab on their computer, or they may be subject-matter experts with no programming experience. It is possible for them to change into cache mode every time they compile the document, but it is cumbersome, and since the files are shared it means that they change the run/cache setting for all other collaborators. In this document we describe how such a group can work efficiently so that the coders can compile in server-mode, and the non-coders will always compile in cache mode.

We recently started working with a publisher and our collaborators there have a premium Overleaf account, which means that they can share files with us via GitHub. However, as of 2022, it is not possible for Overleaf users to install python packages, so our collaborators cannot install the `talk2stat` package, which is needed for `runcode`. This means that when they compile our book, they must do it in cache mode. To make it work, we added a few lines to the preamble and created a make file. Both are described in detail below. The steps below are specific to this example, but can be easily generalized to any similar collaborative project where files are shard by multiple authors.

## The preamble

First, in the preamble of `runcodeOL.tex` we put the following:

```
\IfFileExists{ForceCache}{  
  \usepackage[cache]{runcode}  
}  
{  
  \usepackage[nohup,run,python,stopserver]{runcode}  
}
```

If there is a file called `ForceCache` in the project's directory, the book is compiled in cache mode. Otherwise, we use the server-mode (using Python, in this example.) See more about this in the next section.

We use the `stopserver` option, which is not essential, but we recommend doing it when the project is in its early stages and many changes are made to the code and to the configuration of the server, and the cache files might become outdated.

The `nohup` option is only used because Emacs (for example) always stops all child processes (like `talk2stat`), whether we use `stopserver` or not. The `nohup` option prevents this from happening and then the choice whether to keep the server running or not, is up to us.

Note that we used the `run` option here, but we can still manually change it to `cache` if needed, or override individual invocations of code within the document to be in cache mode.

We also found that when Overleaf compiles the document, temporary files names do not get the main file's prefix. For example, the main file for this example is `runcodeOL.tex` so normally a temporary file created by  $\text{\LaTeX}$  will have `runcodeOL` as a prefix, but this is not the case with Overleaf. To force the prefix to be the main file's name, which is requires so that `runcode` can embed code and results in the pdf, we add the following lines to the preamble of `runcodeOL.tex`:

```
\edef\TeXjobname{\jobname} % (not really essential)
\edef\jobname{\detokenize{runcodeOL}}
```

## The make file

In order to allow us to compile the book and run the code, but allow our publisher to compile the  $\text{\LaTeX}$  file without having to run the code, we created a file called `Makefile` with four options:

- **build**: build the book and use the server-mode.
- **overleaf**: use the cache, compile from tmp files (this must be done after make build, or else the tmp files will not be up to date).
- **clean**: as the name suggests.
- **stopserver**: runs clean, and then stops the `talk2stat` server.

The content of our `Makefile` is this:

```
# Use the cache, compile from tmp files, remove temporary code files
overleaf:
    touch ForceCache
    xelatex -shell-escape runcodeOL.tex

clean:
    rm -f runcodeOL.synctex.gz runcodeOL.aux tmp/*.txt runcodeOL.mw runcodeOL.log

# stop the talk2stat server, but don't compile the book:
stopserver: clean
    python3 -c 'from talk2stat.talk2stat import client; client("./", "python", "QUIT")'
    rm -f serverPIDpython.txt pythondebug.txt talk2stat.log nohup.out

# build the book and use the server, not the cache option:
build: clean
    rm -f ForceCache nohup.out
    xelatex -shell-escape runcodeOL.tex
```

Before pushing the code to GitHub, we run the following sequence of commands in the command line:

```
make stopserver
make build
make overleaf
```

If GitHub is used, we then commit the changes and push to the GitHub server. With Dropbox, OneDrive, etc., the updates will be available to our collaborators without extra steps.

Here is a detailed explanation of the steps:

1. `make stopserver` is used to first remove the `.aux` file and any code files created by `runcode`. It's possible to add to the `clean` option the following suffixes: `.bbl`, `.idx`, `.mw`, `.tbc`, `.toc`, `.mw`, or the cache files, `tmp/*`. Then, it stops the `talk2stat` server, and cleans the files associated with it.
2. `make build` removes the file `ForceCache` in order to ensure that the next compilation will be in server-mode (no cache). Then, the project is compiled (with the mandatory `shell-escape` option).
3. `make overleaf` first creates the `ForceCache` file, so that the next compilation will not invoke the python code. Then, the code is compiled one more time. This is not essential – our collaborators can do it when they pull the latest changes, but it's good to test that the cache-mode compilation worked properly (and save our collaborators the extra step).

## Example

Try compiling this file with the aforementioned steps, and see that after the first compilation the files under the `tmp` folder are updated, but after running `make overleaf` only the main pdf file is updated.

In this example we demonstrate execution of Python code (which computes the Levenshtein distance between two words.)

```
def Levenshtein(s1, s2):
    n = len(s1)
    m = len(s2)
    d = [0] * (n+1)
    for x in range (n+1):
        d[x] = [0] * (m+1)
    for i in range (n):
        d[i+1][0] = i+1
    for j in range (m):
        d[0][j+1] = j+1
    for i in range (n):
        for j in range (m):
            if s1[i] == s2[j]:
                substitutionCost = 0
            else:
                substitutionCost = 1
            d[i+1][j+1] = min(d[i][j+1] + 1,          # deletion
                             d[i+1][j] + 1,          # insertion
                             d[i][j] + substitutionCost) # substitution
    return(d[n][m])

str1 = "sibling"
str2 = "setting"
```

The Levenshtein distance between the words `sibling` and `setting` is 3.