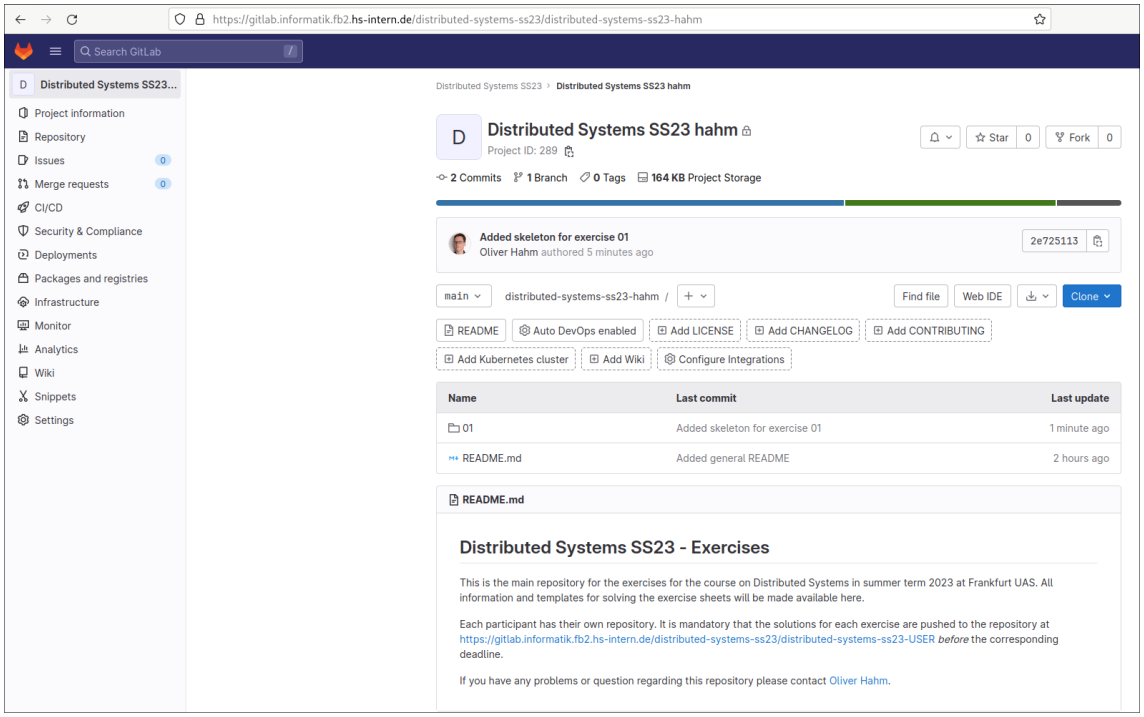
Exercise Sheet 1

Deadline: April 27, 2023 – 04:00 am CEST

Exercise 1 (Clone the Repository)

For the exercises of this course we will work with *git*. The submission for each exercise sheet must be committed to a *git* repository and pushed to a *remote*. In order to do so, you will first have to *clone* your repository from the faculty's *GitLab* instance. The URL for this main repository is:

`https://gitlab.informatik.fb2.hs-intern.de/distributed-systems-ss23/distributed-systems-ss23-<USER>`
(Note: You need to replace ***<USER>*** *with your last name written in small letters and any potential umlauts replaced (ä, ö, ü) ⟶ (ae, oe, ue))*



You can clone your fork from the command line by calling
`git clone <repo>`
(You find the URL for your fork on the GitLab page by clicking on the *Clone* button.)
[1]

It is recommended to use the ssh version. This requires that you create an ssh keypair (using `ssh-keygen`) and upload the public key to GitLab. Go to your GitLab profile (top-right corner) and navigate to *SSH Keys* from the *User Settings* list on the left.

[1] If you get an error about the validity of the TLS certificate, download the server's certificate at `http://teaching.dahahm.de/assets/gitlab-informatik-fb2-hs-intern-de.pem`, and configure git to use it: `git config --global http.sslCAInfo path/to/cert`.

Now copy your public key (per default in */.ssh/id_rsa.pub*), copy-paste it into to the GitLab page, and click on *Add key*.

Once you have successfully cloned your repository, you can start editing the files in your workspace. You can check for local modifications in your workspace by calling `git diff`

In order to commit local changes to the repository **locally**, call
`git add <filename>` and
`git commit`
and set an appropriate *commit message*.

In order to push the local repository upstream to your fork, call
`git push origin main`

ATTENTION: Do not forget this step once your solution is ready for submission! Otherwise your submission cannot be assessed by the lecturer.

Exercise 2 (Work with the Repository)

Create a new file in your repository, save it, add it to git (use `git add <FILENAME>`), commit the change (use `git commit`), and push the changes to the upstream repository (use `git push origin main`). Open the file `README.md` in the editor of your choosing and add the newly created file to the list of files, i.e., add another bullet point under `Directory structure`, list the file name, and describe its content. Save the modifications into the file. Check the local modifications (`git status` and `git diff`) before committing and pushing the changes to the upstream repository (`git add`, `git commit`, and `git push`).

Exercise 3 (Programming C)

We will program in C in the exercises of this course. Even if you have not yet programmed in C, you will probably understand the basics of C rather quickly. You can find multiple books and online tutorials about programming in C, for instance,

- J. Gusted, *Modern C*: `https://modernc.gforge.inria.fr/`
- J. Wolf, *C von A bis Z*: `http://openbook.rheinwerk-verlag.de/c_von_a_bis_z/`

Coding Style

It is expected that you will use a coding style which makes your code readable for other persons. This means in particular that you will use consistent indentations and formatting of your code. It is preferable to put all blocks after `if/else/for/while/...` in curly brackets. You can separate lines that are not coherent in terms of content with blank lines. A useful style guide for readable code can be found, for instance, here:
`https://www.kernel.org/doc/Documentation/process/coding-style.rst`

Error and return code handling

Please note that library functions will report the status of the called operation via their return code. Typically a successful operation will return a zero or a positive number. Errors are usually reported by returning -1. Further error messages may be accessed via the system variable `errno` or by using the `perror` helper function:

```
int fd = open("filename", O_RDONLY);
if (fd == -1) {
    perror("Error on open");
    exit(EXIT_FAILURE);
}
```

Error handling and checking the return values is mandatory. You should always check them.

Hello World!

To warm up to the programming language *C* again, let's start with the classic: implement a small program in C that simply outputs the string "Hello World!". The repository template already provides you with a skeleton file for this, a `Makefile` containing a *rule* to build the program, and a test to verify your success.

It is recommended to use an editor or an IDE (Integrated Development Environment) to work with *C* (or any other) code. On the lab PCs you can, for instance, use *Visual Studio Code* or *Kate* – both offer syntax highlighting for C code. In order to compile (and link) your program you can use the Makefile by calling either simply

```
$ make
```

or by calling make for the specific *target*:

```
$ make helloworld
```

The corresponding test can also be executed via Make by calling

```
$ make test
```

(Note: This will execute the tests for all programs in this exercise, so you will likely get some FAILURE messages.)
You can also execute the specific test manually by calling:

```
$ ./test_helloworld.py
```

Exercise 4 (Working with simple I/O)

In this exercise we will practice handling of command line arguments and file handling via the standard C library. Note the necessary header files as described in the man pages. System calls are documented via man pages in section 2, other library functions can be found in section 3. For example, for the `socket` system call use

```
$ man 2 socket
```

or for the `printf` library call use

```
$ man 3 printf
```

1. Implement a program called `concat` which concatenates strings. The program shall accept two parameters as command line arguments and provide two basic operations:

   - If the first argument is an integer value n, the string of the second argument shall be repeated n times and concatenated (without any whitespaces).
   - Otherwise both arguments shall be handled as strings and concatenated directly to each other.

   The concatenated string shall be printed to *stdout*.

2. Implement a program called `concatfile` which does the same as the program above but stores the output into a file called `concatresults.txt`. The output file shall be created if not existing and overwritten otherwise.

Hint: You can check the results of your programs via the provided test scripts.