Bioinformatics at the DKFZ report

Student: **Ian Dirk Fichtner**
Matriculation nr.: **3404046**

Date: 12th - 13th April 2023
Lecturer: Dr. Lars Feuerbach
Comparative cancer genomics

The aim of this two-day course was to get acquainted with essential bioinformatics tools and concepts like cluster workflow, genome annotation formats, unix, python and bash programming as well as snakemake workflow management. Common mistakes and pitfalls were also tackled.

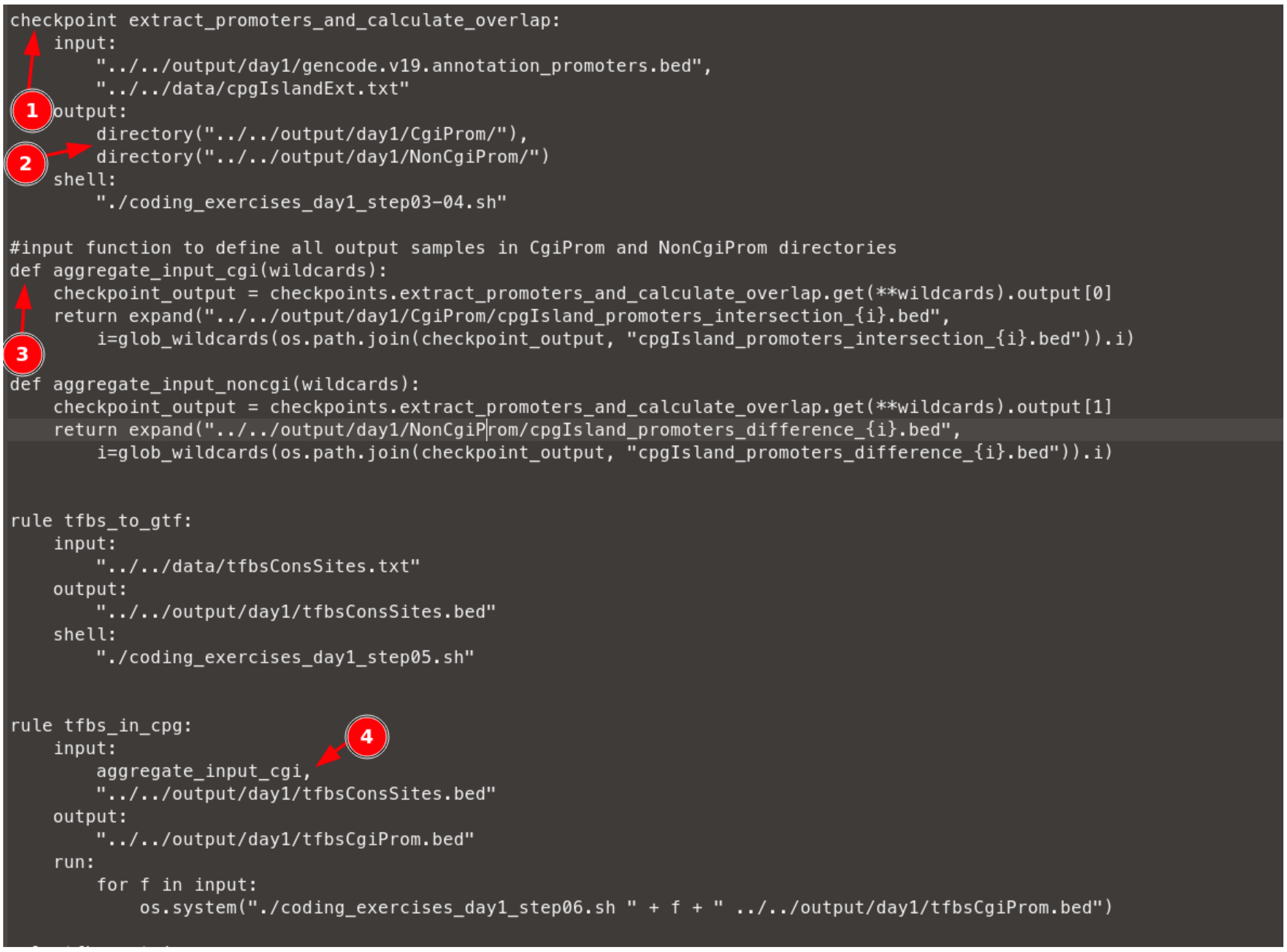
During the first day we had four sessions:

1. Unix, Bash, Python and Bedtools
   We learned basic shell commands, how to use parameters, variables, piping, extracting information from (compressed) text files. Regarding python programming, we learned how to open, write and close files and `sys.stdin/out`. We also learned about the notation of two genome annotation formats: GENCODE (.gtf) and and BED (.bed).
2. Cluster infrastructure
   We learned the cluster interaction workflow which is starts on a client computer (personal pc) which connects via ssh to the worker/submission nodes. From there the jobs are sent to the computing nodes. Basic `qsub` commands were shown and approaches to error handling were presented.
3. Hands-on session - Computer Pool IPMB
   During the evening practicals, the learned concepts were put to test via a specific research question which was solved by using bash, python, R scripting and cluster jobs. In short, we downloaded the GENCODE human genome annotations file, converted it to .bed format extracted relevant information and used bed-tools to analyze TFbs in promoters found in CpG islands.

SOLUTIONS ==> https://github.com/idf-io/bioinformatics-at-the-dkfz
- exercises
- completed pdf script

During the second day, we first got a crash-course in python programming. We then reviewed the hands-on session of the previous day, discussed the answers and further addressed the most common problems. These included annotation file formatting nuances like base-coordinate systems (0- and 1-based), strand handling, chromosome prefixes and the definition of “upstream/downstream” features.

Secondly, we learned about snakemake workflow management programming. Since I was personally very interested in this specific topic, I took the initiative to learn the basics of this framework and apply them to stich together a workflow for the different steps in the hands-on session of the previous day. I learned relevant concepts like how snakemake functions by creating a directed acyclic graph based on the “input” and “output” parameters of the rules and is capable of executing steps in parallel. Snakemake is programming language independent and can use different environments for individual steps. Variable substitution and wildcards provide the real value of this workflow management system. A big point of friction was how to deal with output of variable file number and names, which to use for a next step (rule). I solved this by using `checkpoints` and the `directory` command (see page 2).



```
checkpoint extract_promoters_and_calculate_overlap:
    input:
        "../../output/day1/gencode.v19.annotation_promoters.bed",
        "../../data/cpgIslandExt.txt"
    output:
        directory("../../output/day1/CgiProm/"),
        directory("../../output/day1/NonCgiProm/")
    shell:
        "./coding_exercises_day1_step03-04.sh"

#input function to define all output samples in CgiProm and NonCgiProm directories
def aggregate_input_cgi(wildcards):
    checkpoint_output = checkpoints.extract_promoters_and_calculate_overlap.get(**wildcards).output[0]
    return expand("../../output/day1/CgiProm/cpgIsland_promoters_intersection_{i}.bed",
        i=glob_wildcards(os.path.join(checkpoint_output, "cpgIsland_promoters_intersection_{i}.bed")).i)

def aggregate_input_noncgi(wildcards):
    checkpoint_output = checkpoints.extract_promoters_and_calculate_overlap.get(**wildcards).output[1]
    return expand("../../output/day1/NonCgiProm/cpgIsland_promoters_difference_{i}.bed",
        i=glob_wildcards(os.path.join(checkpoint_output, "cpgIsland_promoters_difference_{i}.bed")).i)


rule tfbs_to_gtf:
    input:
        "../../data/tfbsConsSites.txt"
    output:
        "../../output/day1/tfbsConsSites.bed"
    shell:
        "./coding_exercises_day1_step05.sh"


rule tfbs_in_cpg:
    input:
        aggregate_input_cgi,
        "../../output/day1/tfbsConsSites.bed"
    output:
        "../../output/day1/tfbsCgiProm.bed"
    run:
        for f in input:
            os.system("./coding_exercises_day1_step06.sh " + f + " ../../output/day1/tfbsCgiProm.bed")
```