

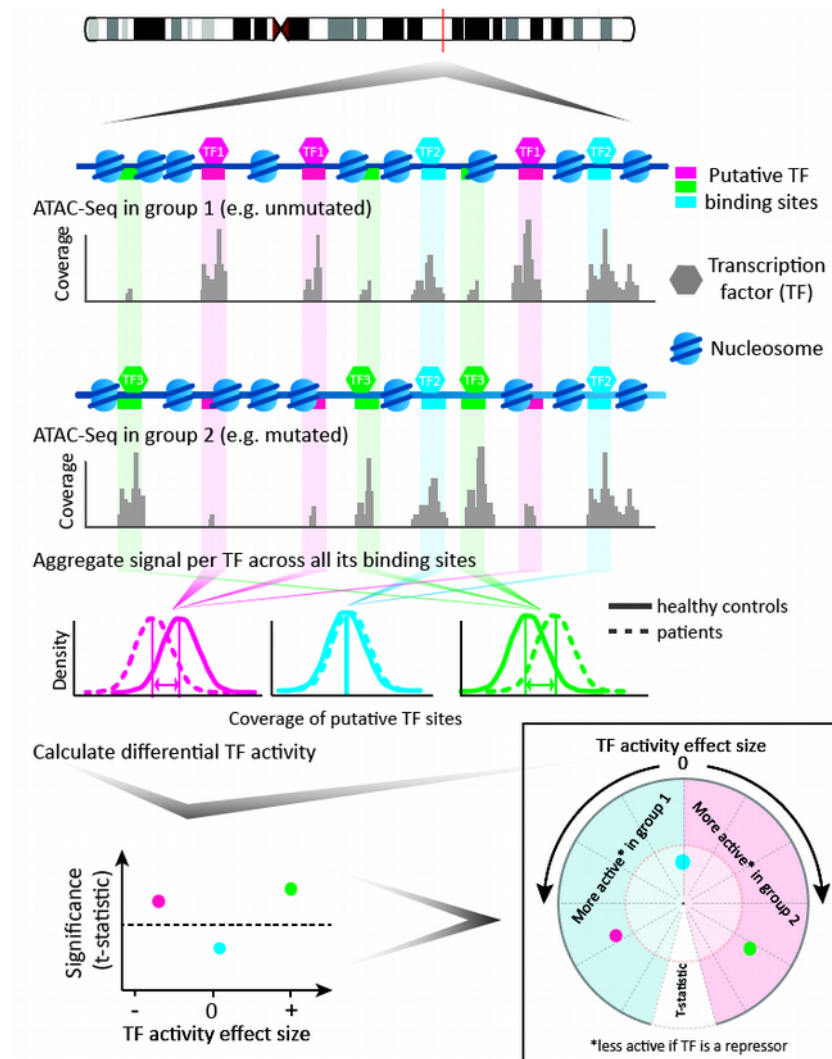
Documentation for

***diffTF*: Genome-wide quantification of differential transcription factor activity**

Ivan Berest¹, Christian Arnold¹, Armando Reyes-Palomares¹ & Judith B. Zaugg¹

¹EMBL-European Molecular Biology Laboratory, Heidelberg, Germany.

Correspondence should be addressed to J.B.Z. (judith.zaugg@embl.de)



Last update: 09/25/17

This document provides documentation and additional information for the *diffTF* tool along with the ATAC-Seq pipeline that we developed independently and in addition.

If you have questions or comments, feel free to contact us. We will be happy to answer any questions related to this project as well as questions related to the software implementation. For method-related questions, contact Judith B. Zaugg (judith.zaugg@embl.de) or Ivan Berest (berest@embl.de). For technical questions, contact Christian Arnold (christian.arnold@embl.de).

If you use this software, please cite the following reference:

Ivan Berest, Christian Arnold, Armando Reyes-Palomares, and Judith B. Zaugg.
diffTF : Genome-wide quantification of differential transcription factor activity. 2017.
in preparation.

Table of Contents

1. Introduction and Availability.....	4
2. Quick Start.....	5
3. General Pipeline Details.....	6
3.1. Prerequisites.....	6
3.1.1. Snakemake.....	6
3.1.2. Wrapper script for Snakemake.....	7
3.1.3. Other software needed by all pipelines.....	7
3.1.4. Other software needed specifically for the <i>diffTF</i> pipeline.....	9
3.1.5. Other software needed specifically for the ATAC-Seq pipeline.....	10
3.2. Preparing a new analysis.....	12
3.3. General files and parameters.....	14
3.3.1. Cluster configuration file for the analysis (cluster.json).....	14
3.3.2. <i>Parameters for calling Snakemake (params.sh)</i>	16
3.3.3. Script to start the analysis and the Snakemake wrapper (runSnakefile.sh).....	21
3.3.4. Input (meta)data.....	21
3.4. Important Notes.....	22
3.4.1. Wrapper script.....	22
3.4.2. Errors during the pipeline.....	22
3.4.3. Performance.....	23
4. <i>diffTF</i> Pipeline: Specific Files and Parameters.....	24
4.1. General configuration file for the analysis (config.json).....	24
4.2. Input metadata (<i>sampleTable.csv</i>).....	28
5. ATAC-Seq Pipeline: Specific Files and Parameters.....	30
5.1. General configuration file for the analysis (config.json).....	30
5.2. Input metadata (<i>sampleTable.csv</i>).....	37

1. Introduction and Availability

We here describe in detail how to run our *diffTF* pipeline for transcription factor binding activity quantification for chromatin accessibility data as well as the independent ATAC-Seq pipeline for proper preprocessing of ATAC-Seq data, both of which are available in the following Github repository: <https://git.embl.de/carnold/TFActivity>.

For a biological motivation, please see the corresponding publication.

Lastly, feel free to contact us, see page 2 for details!

2. Quick Start

The following quick start briefly summarizes the necessary steps to use our pipelines.

1. Install Snakemake (via conda is recommended), see <https://snakemake.readthedocs.io>, and other software needed by our pipelines (see section 3.1)
2. Download the Git repository:

```
$ git clone ...
```
3. To prepare a new analysis, follow the instructions in section 3.2
4. Adjust the analysis-related files accordingly, see sections 3.3 and either 4 or 5.

3. General Pipeline Details

3.1. Prerequisites

First, we have to install a few things. A few command line calls have to be executed hereafter, and they are denoted in the following syntax:

```
$ sh runSnakefile.sh
```

3.1.1. Snakemake

Snakemake is required for running our pipelines. Please ensure that you have at least version 4.0 installed, older versions are not suited.

Principally, there are multiple ways to install Snakemake, two of which we now describe in more detail.

1. Installing Snakemake via miniconda

We recommend installing and maintaining Snakemake via *miniconda*. You should also consider this approach if you want to use the conda-related features in Snakemake for software encapsulation, which we highly recommend. See also

https://snakemake.readthedocs.io/en/stable/getting_started/installation.html for the official documentation of how to install Snakemake.

1. Download the *miniconda* installer

```
$ wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

2. Run the installer

```
$ bash Miniconda3-latest-Linux-x86_64.sh
```

If the installer asks whether or not to append something to your *.bashrc* (a shell script that Bash runs whenever it is started interactively), say yes. If your system does not use *.bashrc* but instead a different file (such as *.profile*), copy the line that the installer added at the end of the *.bashrc* file to the corresponding file on your system.

3. Execute the following lines to add a few channels to your conda:

```
$ conda config --add channels conda-forge
```

```
$ conda config --add channels defaults
```

```
$ conda config --add channels r
```

```
$ conda config --add channels bioconda
```

Now we are ready to install Snakemake. Installing and updating Snakemake and other packages is now trivial:

```
$ conda install snakemake
```

In addition, install a few more packages so that “dot” can be used for Snakemake (see the Snakemake help for details)

```
$ conda install cairo pango graphviz
```

In the future, to update snakemake and other packages, simply type

```
$ conda update snakemake
```

or

```
$ conda update -all
```

2. Installing Snakemake via direct download

Alternatively, download Snakemake directly here:

<https://bitbucket.org/snakemake/snakemake/downloads/>. However, any conda-related features in Snakemake will not be available if conda is not already installed on your system. In addition, updating Snakemake becomes more cumbersome.

3.1.2. Wrapper script for Snakemake

We also need a wrapper script (*runSnakemakeWrapper.sh*) for Snakemake that we developed for convenience reasons. Download it from the Github repository and store it in a directory of your choice.

3.1.3. Other software needed by all pipelines

Other tools are needed to run our pipelines, as outlined below:

1. R and various packages

A working R installation is needed and a number of packages from either CRAN or Bioconductor have to be installed. The combination of packages differs among the various pipelines and are outlined below in the pipeline-specific sections, but the following packages are shared among the various pipelines and are required: "checkmate", "futile.logger", "tidyverse", "reshape2", "tools", "grDevices", "gridExtra", "scales", "rlist". Type the following in R to install them:

```
$ install.packages(c("checkmate", "futile.logger", "tidyverse",  
"reshape2", "gridExtra", "scales", "ggplot2", "jsonlite",  
"RcolorBrewer", "rlist"))
```

2. External tools

Each pipeline needs a number of external tools to work. Depending on whether or not you run the pipeline on your personal computer or a cluster system, setting up these tools might be the most time-consuming part. On a well-maintained cluster system, most if not all of them should already be available since they are commonly used. Independent of the specific pipeline, the following two tools are needed by all and have to be installed and available on the system you run Snakemake from:

1. samtools (*samtools.sourceforge.net*, the executable has to be called *samtools*)
2. bedtools (*http://bedtools.readthedocs.io*, the executable has to be called *bedtools*)

Similarly to how you install Snakemake, you have two ways of installing these tools:

1. Conda-independent installation: Install the tools independent of conda. This reflects the typical way to install new tools on your system. If the tools are already available on your system (type "which samtools" and "which bedtools" to check), then this is the easiest and quickest solution.
2. Conda-dependent installation: If you decide for the conda approach (see also above), you can conveniently install and manage all of these tools centrally with the following command:

```
$ conda install samtools bedtools
```


Note that you can list as many tools as you want, they just have to be space-separated. After successful execution of this command (watch the output if all worked), all of the tools should now be available on your system and you are ready to use our pipeline.

For Snakemake, it is also possible to define isolated software environments per rule. Upon execution of a workflow, the conda package manager is used to obtain and deploy the defined software packages in the specified versions. We highly recommend using this feature for reproducibility, although setting this up properly may take a while. If you want to do so, follow these steps:

1. Generate one or multiple conda environments in which only single tools or a combinations of tools are available and installed. In the following, we demonstrate this by just installing single tools per environment:

```
$ conda create --name samtools python=3 samtools
```

2. Activate the environment export the installed software packages to a yaml file. Change {any_folder} to a folder of your choice in which you store all the conda environment yaml files:

```
$ source activate samtools
```

```
$ conda env export > {any_folder}/envs/samtools.yaml
```

3. Deactivate the environment and reference this yaml file in the central Snakefile for each rule using the tool or combination of tools in the *conda* directive section of the rule:

```
$ source deactivate
```

```
* Adjust the path in the conda: part of each rule*
```

4. Repeat 1 to 3 for the remaining tools if you chose to create one environment per tool (i.e., here, repeat for *bedtools*)

3.1.4. Other software needed specifically for the *diffTF* pipeline

1. R and various packages

A working R installation and the following packages have to be installed in addition to the packages listed above in section 2.1.3: "DESeq2", "vsn", "modeest", "limma", "csaw", "lsr", "dplyr", "ggrepel", "geneplotter", "locfdr", "boot". Type the following in R to install them:

```
$ install.packages(c("ggrepel", "dplyr", "lsr", "modeest",  
"locfdr", "boot"))  
  
source("https://bioconductor.org/biocLite.R")  
biocLite(c("limma", "vsn", "csaw", "DESeq2", "DiffBind",  
"geneplotter"))
```

2. External tools

No further external tools except *samtools* and *bedtools* (see section 2.1.3) are needed.

3.1.5. Other software needed specifically for the ATAC-Seq pipeline

3. R and various packages

A working R installation and the following packages have to be installed in addition to the packages listed above in section 2.1.3: "Rsamtools", "GenomicRanges". Type the following in R to install them:

```
$ source("https://bioconductor.org/biocLite.R")  
  
biocLite(c("Rsamtools", "GenomicRanges"))
```

4. External tools

In addition to the tools listed in section 2.1.3, the following tools have to be installed:

1. Java 8 (needed for GATK and Picard)
2. GATK (<https://software.broadinstitute.org/gatk/>, executed via java, a jar archive is needed which you have to download separately due to license restrictions)
3. Picard (<https://broadinstitute.github.io/picard/>, executed via java, a jar archive is needed which you have to download separately due to licensing issues)
4. FastQC (<https://www.bioinformatics.babraham.ac.uk/projects/fastqc/>, the executable has to be called *fastqc*)

5. Trimmomatic (<http://www.usadellab.org/cms/?page=trimmomatic>, the executable has to be called trimmomatic)
6. Bowtie2 (<http://bowtie-bio.sourceforge.net/bowtie2/index.shtml>, the executable has to be called bowtie2)
7. deepTools (<https://github.com/fidelram/deepTools>, the executable has to be called deeptools, although the individual tool names will be needed, such as plotPCA or correctGCBias)
8. MACS 2 (<https://github.com/taoliu/MACS>, the executable has to be called macs2, see the note below for the Python version)
9. IDR (<https://github.com/nboley/idr>, the executable has to be called idr)
10. MultiQC (<http://multiqc.info/>, the executable has to be called multiqc)

As outlined in section 2.1.3, there are two ways of how to install these tools. We here describe only the one via conda, which is easier and faster. Installing these tools is principally trivial, but there are two minor catches:

1. All tools are compatible with Python3, except MACS2, which only runs under Python 2.7. Assuming that your default conda environment is Python3, you can simply install all tools at once by simply executing the following lines of code in the terminal

```
$ conda install java-jdk fastqc trimmomatic bowtie2 deeptools  
idr multiqc picard gatk
```

MACS2, however, has to be installed separately into a Python 2 environment because each Python environment works only for a particular Python version (that is, 2 or 3), and 2 and 3 are not compatible with one another. The easiest way to do so is via

```
$ conda create --name macs2 python=2.7 macs2
```

To use MACS2 on the command line, simply activate the environment first via

```
$ source activate macs2
```

, then run MACS2, and in the end deactivate it again to return to the default environment

```
$ macs2 ... (run command)
```

```
$ source deactivate macs2
```

To run MACS2 under Snakemake, things become easier: Simply use the conda-directive in each rule that uses MACS2, reference the prepared environment and Snakemake handles the rest!

2. For GATK, due to license restrictions, a few extra steps are needed (see the instructions on <https://bioconda.github.io/recipes/gatk/README.html> for details)

3.2. Preparing a new analysis

To prepare a new analysis (hereafter named *TEST_ANALYSIS*), follow the following steps:

1. Create a new folder *TEST_ANALYSIS* in a location of your choice.
2. Create two folders *INPUT* and *OUTPUT* within *TEST_ANALYSIS*.
3. In the *INPUT* folder, copy the following files from a previous analysis or from the *TEMPLATES* folder in the Github repository:

1. *config.json*
2. *sampleTable.csv*
3. *cluster.json*
4. *params.sh*
5. *runSnakefile.sh*

In addition, for data organization and management reasons, we also advise to put the actual input data for the analysis in a *data* subfolder.

Except for *params.sh* (the name of which can be adjusted in the *runSnakefile.sh* file), the exact names of these five files do not matter as long as they are correctly specified in the corresponding files in which they are defined. Feel free to modify them according to your needs.

The reason for such a, at first sight, complicated setup is that it greatly facilitates managing and altering parameters and clearly separating analysis-associated and Snakemake-associated parameters. You will quickly see that it becomes very easy to work with this setup, as is it identical across analyses and very flexible, thereby also increasing reproducibility and decreasing development time.

4. Modify the files *cluster.json*, *config.json*, *params.sh* and *sampleTable.csv* according to your needs. See the next section for parameter details.
5. For the first run, it is highly recommended to make sure that you run Snakemake in dry mode to see what it would actually do and execute. For this, set the *dryRun* parameter in *params.sh* to TRUE and start the Snakemake dry run by typing

```
$ sh runSnakefile.sh
```

If you rerun only parts of an analysis, make sure that Snakemake does not recompute parts that are already present, this sometimes happens for various reasons.

6. If you do not receive any warnings or error messages and Snakemake lists the jobs that it would compute, change *dryRun* back to FALSE and start the analysis again via

```
$ sh runSnakefile.sh
```

Note that this command eventually also submits jobs automatically (if run in a cluster environment). There is no need to manually call *bsub* or any architecture-specific commands. If the expected running time of the analysis is high, you have to think about how to best run Snakemake. These are the main options:

1. Execute Snakemake simply on the terminal. This is the easiest approach, but with one important disadvantage: If you set your PC in sleep mode or shutdown, the analysis is also interrupted.
2. Use a method to allow Snakemake to run in the background independent of the state of your PC. The easiest approach might be *screen*. It is easy to learn and preinstalled on most systems. Alternatively, put the Snakemake job in the background, but then you won't see the output and the current state of the analysis (see <https://stackoverflow.com/questions/625409/how-do-i-put-an-already-running-process-under-nohup>) unless you decided to redirect the Snakemake output to a file via "> *Snakemake_output*" or something alike.

3.3. General files and parameters

We now describe the various files that need to be present to start any of the analyses pipeline we describe. As outlined above, we advise to copy and modify from the TEMPLATE folder in the Github repository.

3.3.1. Cluster configuration file for the analysis (cluster.json)

This file in JSON format is only needed if you run the analysis in a cluster environment. If you run an analysis locally, you can ignore this file.

Currently, our wrapper script supports both LSF/bsub and SLURM (see the *params.sh* section for details). We cannot explain all technical details here, please check the Snakemake documentation and our examples for orientation. The provided *cluster.json* templates for each pipeline work flawlessly for us and should also work for you. If your cluster system is neither LSF nor SLURM, the cluster config has to be modified, which should however be quick and easy because the same principle applies for other systems as well and Snakemake makes it already easy to abstract from the specific architecture.

If LSF/bsub is used, the cluster.json supports the following parameters in the `__default__` section:

- queue (the queue name where jobs should be transmitted to. Default: *medium_priority*. Adjust to the queue name of your choice.)
- name (an arbitrary name for each job. Wildcards are supported. The default name we use is `analysisName.{rule}.{wildcards}`. Replace `analysisName` by any name of your choice)
- resources (additional memory-related resources specific for the system. Our default is `"select[mem>10000] rusage[mem=10000] span[hosts=1]"`. Note the “\” which are important!
- memory (requested default memory. We suggest a default of 10000 – that is, 10 GB)
- output (path to the output file. Wildcards are supported. A reasonable default name here is, for example, `{rule}.{wildcards}.out`)
- error (path to the error file. Wildcards are supported. A reasonable default name here is, for example, `{rule}.{wildcards}.err`)

The wrapper script transforms these values into the following syntax that is passed on to Snakemake: `bsub -q {cluster.queue} -J {cluster.name} -n {threads} -R \"{cluster.resources}\" -M {cluster.memory} -o \"{cluster.output}\" -e \"{cluster.error}\"`

If SLURM is used, the cluster.json has to contain the following parameters in the `__default__` section:

- `queueSLURM` (the queue name where jobs should be transmitted to. Default: 1day. Adjust to the queue name of your choice.)
- `group` (the name of the group you are in)
- `name` (an arbitrary name for each job. Wildcards are supported. The default name we use is `analysisName.{rule}.{wildcards}`. Replace `analysisName` by any name of your choice)
- `memory` (requested default memory. We suggest a default of at least 20000)
- `maxTime`: The maximum allowed running time. Depending on the complexity of the analyses, individual steps may need multiple hours to finish.
- `output` (path to the output file. Wildcards are supported. A reasonable default name here is, for example, `{rule}.{wildcards}.out`)
- `nNodes`: Number of nodes that are allocated for this job. Corresponds to the `-N` option of `sbatch` (Default: 1).
- `nCores`: Corresponds to the `-n` option of `sbatch` (Default: 16)
- `error` (path to the error file. Wildcards are supported. A reasonable default name here is, for example, `{rule}.{wildcards}.err`)

The wrapper script transforms these values into the following syntax that is passed on to Snakemake: `sbatch -p {cluster.queueSLURM} -J {cluster.name} -A {cluster.group} -N {cluster.nNodes} -n {cluster.nCores} --mem {cluster.memory} -o \"{cluster.output}\" -e \"{cluster.error}\" --mail-type NONE \"`

Feel free to extend this to suit your needs by adding parameters to the cluster config and editing the wrapper script. See also <https://slurm.schedmd.com/sbatch.html> for details.

If you run the analysis on either LSF or SLURM, the changes to make are minimal:

1. Adjust the first part of the output and error files (everything before the `Logs_and_Benchmarks` folder) and replace the absolute path according with the value of the

outdir parameter as specified in the *config.json* file. The *Logs_and_Benchmarks* part is automatically created by the wrapper script, do not delete this.

2. Memory requirements for the various rules have been set generously so that it also works for larger amounts of data. There is principally no need to change these values unless you want to decrease the memory footprint further. The values in the *__default__* section apply to all rules unless overwritten explicitly below.
3. Optional: Adjust the name of the analysis so that it becomes easier for you to track the analysis status on the cluster if multiple analyses are run simultaneously.

3.3.2. Parameters for calling Snakemake (*params.sh*)

This file defines Snakemake-specific parameters only, all of which are explained in the following. Parameters in **bold** are the most important ones and the ones that usually have to be modified, while non-bold indicates parameters that only have to be touched in particular cases. Parameters in *gray* are either obsolete or do not have to be touched unless there is a very specific reason.

| **configFile** |

String. Default “config.json”. Corresponds to *--configfile* in Snakemake.

Path to the config file. Since the config file is usually located in the same directory, this can be left untouched unless it has been renamed or moved.

| **snakefile** |

String. Default “Snakefile”. Path to the Snakefile. Corresponds to *--snakefile* in Snakemake.

The Snakefile to run. Adjust this accordingly.

| **nCores** |

Integer > 0. Default 5. Corresponds to *--cores* in Snakemake.

Maximum number of CPUs per rule (if the rule supports parallel execution). Make to set this to reasonable values. If Snakemake is executed in a cluster environment, this value is currently force set to 16. Only relevant for rules with threads directive values of > 1.

| **dryRun** |

Logical. TRUE or FALSE. Default TRUE. Corresponds to *--dryrun* in Snakemake.

Run in dry mode without actually computing something? See the Snakemake help for *--dryrun* for more details.

| **submitToCluster** |

Logical. TRUE or FALSE. Default FALSE. Corresponds to *--cluster* in Snakemake.

Execute Snakemake rules with the given submit command e.g. bsub or SLURM? Set to TRUE to run the analysis on the cluster. If set to TRUE, a cluster config file (cluster.json) has to be specified. See the Snakemake help for *--cluster* and the corresponding section for cluster.json above for more details.

| **clusterConfig** |

String. Path to the cluster config. Default "cluster.json". Only relevant if *submitToCluster* is set to TRUE.

Path to the cluster config file (relative path suffices usually) that defines the wildcards used in 'cluster' for specific rules, instead of having them specified in the Snakefile. See the cluster config section for more details.

| **useConda** |

Logical. TRUE or FALSE. Default FALSE. Corresponds to *--use-conda* in Snakemake.

Should Snakemake use conda and run each rule with a conda directive in its own isolated environment? See the Snakemake help for *--use-conda* for more details.

| **condaDir** |

String. Path to any directory. Default "" (empty string). Corresponds to *--conda-prefix* in Snakemake.

Specify a directory in which the 'conda' and 'conda-archive' directories are created. These are used to store conda environments and their archives, respectively. If not supplied, the value is set to the '.snakemake' directory relative to the invocation directory. See the Snakemake help for *--conda-prefix* for more details.

| forceRerunAll |

Logical. TRUE or FALSE. Default FALSE. Corresponds to *--forceall* in Snakemake.

Forces to rerun the whole pipeline even if the output files are already present. See the Snakemake help for *--forceall* for more details.

| ignoreTemp |

Logical. TRUE or FALSE. Default TRUE. Corresponds to *--notemp* in Snakemake.

If set to TRUE, temporary files (as declared by the temp directive) are not deleted. This might be a reasonably good idea in the beginning to check if the pipeline runs through. Also, this is useful when running only a part of the workflow since temp() may lead to deletion of files required by other parts of the workflow. To save disk space, set to FALSE. Advanced option. See the Snakemake help for *--notemp* for more details.

| touchOutputFiles |

Logical. TRUE or FALSE. Default FALSE.

Don't run Snakemake, just update the time stamps for output files. This is sometimes useful when you manually mess with the timestamps of the output files outside of Snakemake. Advanced option.

| allowedRules |

String. Name of rule(s), separated by space. Default "" (empty string): option disabled. Corresponds to *--allowed-rules* in Snakemake.

Which rules are allowed to run? If you want to restrict which rules are allowed to run, name the rules as they appear in the Snakefile, separated by spaces. Advanced option. See the Snakemake help for *--allowed-rules* for more details.

| runSpecificRule |

String. Name of rule. Default "" (empty string): option disabled. Corresponds to *--forcerun* in Snakemake.

Force the re-execution or creation of the given rules or files. Name the rule as it appears in the Snakefile. Advanced option. See the Snakemake help for *--forcerun* for more details.

| rerunIncomplete |

Logical. TRUE or FALSE. Default TRUE. Corresponds to *--rerun-incomplete* in Snakemake.

Re-run all jobs the output of which is recognized as incomplete. See the Snakemake help for *--rerun-incomplete* for more details.

| runAlsoDownstreamRules |

Logical. TRUE or FALSE. Default TRUE. Only relevant if *runSpecificRule* is set to a non-default value.

If set to FALSE, ONLY the rule as specified in *runSpecificRule* will be run (internally, the *--until* option is appended after *--forcerun* with the same rule); otherwise, all downstream rules are also triggered. Advanced option. See the Snakemake help for *--until* for more details.

| useSLURM |

Logical. TRUE or FALSE. Default FALSE.

If set to TRUE, preconfigures Snakemake for SLURM systems using *sbatch*. If set to FALSE, preconfigures Snakemake for LSF systems using *bsub*. See the Snakemake help for *--cluster* for more details. If SLURM is used, make sure to adjust the *cluster.json* accordingly.

| maxJobsCluster |

Integer > 0 & < 500. Default 200. Only relevant if *submitToCluster* is set to TRUE. Corresponds to *--jobs* in Snakemake.

The maximum number of simultaneous jobs that are allowed to run. Be careful with this option and do not set it to higher values unless you know the IO and general performance of the underlying system is not noticeably influenced. See the Snakemake help for *-- jobs* for more details.

| maxRestartsPerJob |

Integer >= 0. Default 1. Corresponds to *--restart-times* in Snakemake.

The number of times a job that fails should be re-executed. Increase to higher values (we recommend 2) for unstable cluster systems. See the Snakemake help for *-- restart-times* for more details.

| nlock |

Logical. TRUE or FALSE. Default TRUE. Corresponds to *--nlock* in Snakemake.

Do not lock the working directory when set to TRUE, which is the default. See the Snakemake help for *-- nlock* for more details.

| useVerbose |

Logical. TRUE or FALSE. Default FALSE. Corresponds to *--verbose* and *--printshellcmds* in Snakemake.

Print debugging output. If set to TRUE, also *--printshellcmds* is added.

| runCustomCommand |

String. Default "" (empty string).

When set to a non-default value, this option enables you to run a custom command instead any automated command. You have to write the full manual command, any other specified options will not (!) be considered. Advanced option.

| skipSummaryAndDAG |

Logical. TRUE or FALSE. Default TRUE.

Should the workflow graph be produced (invoking “dot”)?As this may need a long time for complex analyses, we strongly recommend setting this value to TRUE and running that separately at a later point. See the Snakemake help for further details.

| workflowGraphFileType |

String. Either “pdf” or “svg” or ”png”. Default “pdf”.

Ignored if skipSummaryAndDAG is set to TRUE (the default). You can usually leave this untouched unless you also suffer from the "bug" that the program dot has a problem producing PDF files (then change to "svg" or "png")

3.3.3. Script to start the analysis and the Snakemake wrapper (runSnakefile.sh)

For this short script, only the absolute path to the wrapper script that has been downloaded before has to be modified once. If *params.sh* has been renamed, the new name has to be adjusted in this file also.

3.3.4. Input (meta)data

This file summarizes the data and corresponding metadata that are available and that should be used for the analysis. The format is flexible and may contain additional information that is currently ignored by the pipeline, so it should be used to capture all the available information in a single place. Importantly, the file must be saved as tab-separated, the exact name does not matter as long as it is correctly specified in the configuration file. The required columns differ among the pipelines, check the details in the corresponding sections below.

3.4. Important Notes

3.4.1. Wrapper script

In addition to the options in the Snakemake params file, the following Snakemake-related options are currently always invoked by the wrapper script: `--timestamp`, `--keep-going`, `--reason`, `--latency-wait 30`, `--stats`.

Although the wrapper script supports a variety of Snakemake-related options, it has also limitations. Options not mentioned in the `params.sh` file above are currently not supported by the wrapper script. This includes, for example, the usage of particular advanced Snakemake options or cluster-related options such as using DRMAA. Feel free to extend the wrapper script to suit your needs!

For integrity reasons, the wrapper script currently searches for empty output files before calling Snakemake. The reason for this is that usually, empty files denote that something went wrong beforehand. If you modified our Snakefile and you know that you have and need empty output files, feel free to comment the part out of the wrapper script.

Since the `--nolock` parameter is enabled by default, **do not attempt to run multiple Snakemake analyses in the same folder!** Unless you changed the `--nolock` option, this will not immediately result in an error, but it will corrupt the metadata files that Snakemake produces.

3.4.2. Errors during the pipeline

Errors occur during the Snakemake run can principally be divided into:

- Temporary errors
 - might occur due to temporary problems such as bad nodes, file system issues or latencies
 - rerunning usually fixes the problem already
 - The default Snakemake configuration is to rerun failing jobs automatically (twice at most, see the parameter list in the wrapper script for details), which usually eliminates temporary problems almost completely. Feel free to change this value accordingly.
- Permanent errors
 - indicates a real error related to the specific command that is executed
 - rerunning does not fix the problem as it is systematic

To troubleshoot errors, you have to first locate the exact error. Depending on how you run Snakemake (i.e., in a cluster setting or not), check the following places:

- in locale mode: the Snakemake output on the console
- in cluster mode: the error OR output file of the corresponding rule that threw the error. In the default setup, the error file is named identically to the rule and has a “.err” and “.out”, respectively, in the end of the filename, along with wildcard assignments for easier identification. Check the “Logs_and_Benchmarks” directory

After locating the error, fix it accordingly. For example, if you receive a memory-related error, try to increase the available memory. In a cluster setting, for example, adjust the cluster file accordingly by either increasing the default memory or (preferably) adding the rule that threw an error to the list of exceptions and override the default values.

After fixing the error, simply rerun Snakemake via “sh runSnakefile.sh”.

3.4.3. Performance

- Since running the pipeline might be computationally demanding, make sure you have enough space left on your device. Preferably, use a cluster system and put the folder in a file system with high IO performance.
- if the number of jobs in parallel is high, system IO might increase substantially
- running times and memory requirements somewhat scale with the number of permutations. If you specify a high number of permutations (> 5 let's say), adjust the time and memory settings accordingly.
- If you have to abort a current Snakemake run, press CTRL+C once (!) and then manually kill all running jobs associated with this analysis. Snakemake does not kill jobs automatically if you abort it unless you use it in the `--cluster-sync` or `--drmaa` mode.

4. *diffTF* Pipeline: Specific Files and Parameters

As outlined in section 2.2, five different files have to be present for any new analysis, three of which share the same content across analyses and have already been described in section 2.3. We now describe the remaining two files that contain content and parameters specific to the *diffTF* pipeline.

4.1. General configuration file for the analysis (config.json)

Section “par_general”

outdir

String. Default “output”. Root output directory.

The root output directory where all output is stored.

regionExtension

Integer > 0. Default 100. Target region extension in base pairs.

This parameter specifies the number of base pairs each target region (from the peaks file) should be extended in both 5’ and 3’ direction.

comparisonType

String. Default ‘’.

This parameter helps to organize complex analysis for which multiple different types of comparisons should be done. Set it to a short but descriptive name that summarizes the type of comparison you are making or the types of cells you compare. The value of this parameter appears as prefix in most output files created by the pipeline. It may also be empty.

conditionComparison

String. Default “mutated,unmutated”. Two conditions to compare against.

This defines the two (!) conditions that should be compared in the TF activity pipeline. The names must be comma-separated and present in the corresponding column in the metadata file. Only those samples corresponding to one of the two specified conditions will be taken as input. If a sample has another, third condition, it will therefore be ignored.

designContrast

String. Default "~ Treatment + Condition". Design formula.

The variable corresponding to the two conditions comes typically last (here: Condition).

designVariableTypes

String. Default "Treatment:factor, Condition:factor".

The data types of all elements listed in *designContrast*. Names must be separated by commas, spaces are allowed and will be eliminated automatically. The data type must be specified with a ":", followed by either "numeric", "integer", "logical", or "factor". The variable corresponding to the condition must (!) come last!

nPermutations

Integer ≥ 0 . Default 0. The number of random sample permutations.

If set to a value > 0 , in addition to the real and non-permuted data, the sample conditions as specified in the sample table and in the parameter *conditionComparison* will be randomly permuted *nPermutations* times while retaining the original frequencies of the two conditions. The full pipeline is then run independently for each permutation also. In the final circular visualization, additional 5 and 95% thresholds are then drawn to mark the region of the plot that cannot be distinguished from noise, therefore increasing accuracy of the results. Note that the running time of all R scripts will be increased, currently only the "prepareData" R script uses multiple CPUs, the others will calculate the necessary intermediate results sequentially for each permutation, thereby increasing running time linearly with the number of permutations.

limit_nTF_to

Integer ≥ -1 . Default -1. Run the analysis only for a subset of the Tfs for testing purposes.

If set to a value > 0 , only the specified number of Tfs are analyzed. Use this parameter to test if the pipeline works by setting it to a small value (say 10) initially. If all runs through, change it back to -1.

dir_scripts

String. Default "". The path to the directory where the R scripts for running the pipeline are stored.

RNASeqIntegration

Logical, true or false. Default false. Should RNA-Seq data be integrated into the pipeline?

If set to true, RNA-Seq counts as specified in the parameter *RNASeqCounts* will be used to classify each TF into either “activator”, “repressor”, “unknown”, or “not-expressed” for the final circular visualization and the summary table. For methodological details, see the Methods details of the publication.

Section "samples"

summaryFile

String. Default ‘samples.tsv’. Path to the sample metadata spreadsheet.

Path to a tab-separated file that summarizes the input data. See the next section and the example file for how this file should look like.

Section "peaks"

consensusPeaks

String. Default “ (empty)”. Path to the consensus peak file. If set to the empty string, the pipeline will generate a consensus peaks out of the peak files from each individual sample

If no file is provided, the pipeline computes consensus peaks based on all individual peak files. For this, you need to provide the following two things:

- a peak file for each sample in the metadata file in the column “peaks”, see the next section for details.
- The format of the peak files, as specified in the parameter *peakType* (see below)

If a file is provided, it has to match exactly(!) the following specifications:

- tab-separated columns
- no column names in the first row
- five columns in total:
 1. Chromosome
 2. Start position

3. End position
4. Identifier
5. Line number (starting from 1)

peakType

String. Default 'narrow'. Format of the peaks. Only relevant if no consensus peak file has been provided.

Only needed if no consensus peak set has been provided. Currently, all individual peak files must be in the same format. Because we use *DiffBind* to create the consensus peaks, the following formats are supported (taken from the *DiffBind* help for *dba*):

- "raw": text file file; peak score is in fourth column
- "bed": .bed file; peak score is in fifth column
- "narrow": default peak.format: narrowPeaks file
- "macs": MACS .xls file
- "swembl": SWEMBL .peaks file
- "bayes": bayesPeak file
- "peakset": peakset written out using `pv.writepeakset`
- "fp4": FindPeaks v4

minOverlap

Integer ≥ 0 or Float between 0 and 1. Default 2. Minimum overlap for peak files for a peak to be considered into the consensus peak set. Corresponds to the *minOverlap* argument in the *dba* function of *DiffBind*. Only needed when *consensusPeaks* is not provided.

Only include peaks in at least this many peak sets in the main binding matrix. Only needed when *consensusPeaks* is not provided. If *minOverlap* is between zero and one, peak will be included from at least this proportion of peaksets. For more information, see the *minOverlap* argument in the *dba* function of *DiffBind*.

Section "additionalInputFiles"

refGenome fasta

String. Default 'hg19.fasta'. Path to the reference genome FASTA file.

The path to the reference genome file in FASTA format.

dir PWMScan

The directory where the TF-specific PWM results are stored, in BED format. Each TF has to have one bed file, in the format $\{TF\}.bed$. For your convenience, we already provide all files for the TF that we used for your analyses, a total of 620 human and 423 mouse TF (see *PWMscan.human.tar.gz* and *PWMscan.mouse.tar.gz*). However, you may also manually create these files for additional TF of your choice. For this, see the HOCOMOCO manual).

RNASeqCounts

String. Default "". Path to the file with RNA-Seq counts.

If the parameter *RNASeqIntegration* is set to true, a tab-separated file has to be specified with normalized RNA-Seq counts. The first line must be used for labeling the samples, with column names being identical to the sample names as specific in the sample summary table (parameter *summaryFile*). The first column must be named ENSEMBL and it must contain ENSEMBL IDs (e.g., ENSG00000028277) without dots. The IDs are then matched to the IDs as specified in the *HOCOMOCO_mapping* file.

HOCOMOCO mapping

String. Default "HOCOMOCO/HOCOTFID2ENSEMBL.txt". Path to the file with the HOCOMOCO mapping.

If RNA-Seq integration shall be used, a translation table to associate Tfs and ENSEMBL genes is needed. For convenience, we provide such a translation table for human and mouse. If you want to use your own version, check the example translation tables and construct one that has an identical structure.

4.2. Input metadata (*sampleTable.csv*)

In addition to the information mentioned above (general notes about this file and requirements), the structure of this file has to be as follows:

- It must contain at least contain the following columns (the exact names do matter):
 - "sampleID": The ID of the sample

- “*bamReads*”: path to the BAM file corresponding to the sample. Note that the BAM files must be valid BAM files with chromosome names that have a “chr” as prefix. The pipeline may crash if the “chr” part is missing.
- “*conditionSummary*”: particular condition the samples belongs to. Note that while more than two groups can be defined (e.g., disease-mutated, disease-unmutated, wildtype) across samples, only two specific groups can be compared for each analysis.
- All variable names from the design formula must also be present as a separate column with reasonable input data

5. ATAC-Seq Pipeline: Specific Files and Parameters

As outlined in section 2.2, five different files have to be present for any new analysis, three of which share the same content across analyses and have already been described in section 2.3. We now describe the remaining two files that contain content and parameters specific to the ATAC-Seq pipeline.

5.1. General configuration file for the analysis (config.json)

Section "par_general"

|outdir|

STRING. Output directory. Will be created if not yet present. Specify absolute path here.

Example: "/g/scb2/zaugg/carnold/Projects/AtacSeq/example/output2"

|pairedEnd|

BOOLEAN. true or false. Default true. Paired end data?

Single-end ATAC-Seq data is not yet supported with this pipeline and it requires rewriting parts of it. If set to "false", the Snakemake pipeline will abort in the beginning.

Section "additionalInputFiles"

|trimmomatic_adapters|

STRING. Absolute path to the adapters file for Trimmomatic in .fa format.

|blacklistRegions|

STRING. Absolute path to a BED file that contains the genomic regions that should be filtered from the peaks

The following three files are required for base quality score recalibration using GATK "BaseRecalibrator". See the help pages of the tool "BaseRecalibrator" from GATK for more details. We recommend downloading the GATK_bundle, which contains all necessary files.

|knownSNPs|

STRING. Absolute path to a database of known polymorphic sites (SNPs). Supported formats from GATK: BCF2, BEAGLE, BED, BEDTABLE, EXAMPLEBINARY, GELITEXT, RAWHAPMAP, REFSEQ, SAMPILEUP, SAMREAD, TABLE, VCF, VCF3

|knownIndels|

STRING. Absolute path to a database of known polymorphic sites (Indels). See description above.

|refGenome_fasta|

STRING. Absolute path to a fasta file with the reference genome. Importantly, note that this has to correspond to the same genome assembly version as the alignment as well as the database of polymorphic sites as specified above.

"refGenome_dict":

"/g/scb2/zaugg/zaugg_shared/annotations/hg19/GATK_bundle/ucsc.hg19.onlyRefChr.dict",

"refGenome_2bit :

"/g/scb2/zaugg/zaugg_shared/annotations/hg19/GATK_bundle/ucsc.hg19.2bit",

"annotationGTF" :

"/g/scb2/zaugg/zaugg_shared/annotations/hg19/Genome_v19/genome.v19.annotation.gtf"

Section "executables"

|java_exec|

STRING. (Path to the) java executable. Java version must be at least 1.8!

"GATK_jar"

"/g/scb2/zaugg/carnold/Projects/AtacSeq/src/Snakemake/tools/GenomeAnalysisTK.jar",

"PICARD_jar"

STRING. (Path to the) picardtools main jar file

Example: "/g/scb2/zaugg/zaugg_shared/Programs/Picardtools/picard.jar"

|STATS_script|

STRING. (Absolute path to the) STATS script path (R script)

Example: "/g/scb2/zaugg/carnold/Projects/AtacSeq/src/Snakemake/src/aut_stats.R"

|FL_distr_script|

STRING. (Absolute path to the) FL_distr script path (R script)

Example: "/g/scb2/zaugg/carnold/Projects/AtacSeq/src/Snakemake/src/Fragment_length.R"

Section "par_trimming"

|trimmomatic_ILLUMINACLIP|

STRING. ILLUMINACLIP value. See trimmomatic manual

Example: "1:30:4:5:true"

|trimmomatic_trailing|

INTEGER. TRAILING value. See trimmomatic manual

Example: 3

|trimmomatic_minlen|

INTEGER. MINLEN value. See trimmomatic manual

Example: 20

|trimmomatic_phredType|

STRING. Phred type. See trimmomatic manual. The "-" is added automatically by the Snakemake pipeline.

Example: "phred33"

Section "par_align"

|bowtie2_sensitivity|

STRING. Sensitivity. Leave empty for the default sensitivity. See bowtie2 manual.

Example: "--very-sensitive"

|bowtie2_X|

INTEGER. Value for parameter X. See bowtie2 manual.

Example: 2000

|bowtie2_refGenome|

STRING. Value for parameter x. See bowtie2 manual.

Example: "/g/scb/zaugg/zaugg_shared/annotations/hg19/referenceGenome/Bowtie2/hg19"

|assemblyVersion|

STRING. Reference genome assembly version. Must match the one used by the alignment program.

Example: "hg19"

Section "par_postalign"

|minMAPQscore|

INTEGER. Minimum MAPQ score. Reads with a lower MAPQ quality will be removed during the processing.

Example: 10

|ValidationStringencySortSam|

STRING. Value of the VALIDATION_STRINGENCY from SortSam (Picard tools). See the manual for details.

Example: "LENIENT"

|ValidationStringencyMarkDuplicates|

STRING. Value of the VALIDATION_STRINGENCY from MarkDuplicates (Picard tools). See the manual for details.

Example: "SILENT"

|CIGAR|

STRING. Used for filtering reads (default: filter INDELS). Relates to the one letter abbreviations for CIGAR strings such as I for insertion and D for deletion.

Specify all the one letter abbreviations in the CIGAR string of a read here that should be filtered.

"ID" would keep a read only if the CIGAR string does not contain the letters "I" and "D" (e.g., only M for example)

Example: "ID"

|adjustRSS_forward|

INTEGER. Adjustment of the read start positions on the forward strand. Should be a positive number. See the Buenrostro paper for details.

Example: 4

|adjustRSS_reverse|

INTEGER. Adjustment of the read start positions on the reverse strand. Should be a negative number. See the Buenrostro paper for details.

Example: -5

Section "par_scripts"

The STATS script does a TSS enrichment test to test whether or not ATAC-Seq reads are primarily located within annotated TSS as opposed to outside of TSS regions.

|STATS_script_withinThr|

INTEGER. The region size in bp that specifies what is considered within a TSS. A value of 4000 means the region from -2kb up to +2kb of annotated TSS.

Example: 4000

|STATS_script_outsideThr|

INTEGER. The size of the region adjacent to the within TSS region that is considered outside of a TSS.

A value of 1000 therefore denotes the 1kb region up- and downstream of the within TSS region (from -3 to -2kb upstream and from +2 to +3 kb downstream of annotated TSS.)

Example: 1000

|STATS_script_geneTypesToKeep|

STRING. Gene type to keep / do the analyses for. Allowed are gene types as specified by GENCODE. The default is "protein_coding".

Example: "protein_coding"

|FL_distr_script_cutoff|

INTEGER. Fragment length cutoff. All reads with a fragment length less than this value will be filtered for the purpose of this script.

Example: 600

Section "par_peakCalling"

```
"modelNonStringent": "--nolambda --nomodel",  
"modelStringent": "--nomodel",  
"modelStringent_minQValue" : 0.01,  
"modelNonStringent_minQValue" : 0.1,  
"modelNonStringent_slocal": 10000,  
"Encode_pValThreshold": 0.1,  
"Encode_modelBroadAndGapped": "--nomodel --shift -75 --extsize 150 --broad --keep-dup all",  
"Encode_modelNarrow": "--nomodel --shift -75 --extsize 150 -B --SPMR --keep-dup all --call-summits"
```

Section "par_deepTools"

|effectiveGenomeSize|

|bamCoverage_normalizationCoverage|

STRING. Either "normalizeTo1x NUMBER" OR "normalizeUsingRPKM" (note the missing (!) leading "--"), where NUMBER denotes. Beware of the mapping dependence on the read length: The reported numbers on the websites are for 30bp reads, and we now have much longer reads usually. See Koehler et al. (2011) for numbers (Koehler, R., Issac, H., Cloonan, N., & Grimmond, S. M. (2011). The uniqueome: a mappability resource for short-tag sequencing. *Bioinformatics*, 27(2), 272-274.)

Example: "normalizeUsingRPKM 2487768882" for hg19/hg38 and 75bp reads

|bamCoverage_binSize|

INTEGER. Size of the bins, in bases

Example: 10

|bamCoverage_otherOptions|

STRING. Additional options that are supported by bamCoverage. Note that the "--" or "-" has to be present here.

Example: "--extendReads"

5.2. Input metadata (*sampleTable.csv*)

In addition to the information mentioned above (general notes about this file and requirements), the structure of this file has to be as follows:

- It must contain at least contain the following columns (the exact names do matter):
 - *'individual'*: The name of the individual the sample belongs to.
 - *'sampleName'*: The name of the sample. Must correspond 1:1 to the corresponding file names except the suffix ("_1" and "_2", see below). Thus, if the sample names are denoted as "test1_rep1" and "test2_rep1", the corresponding files in the input folder must be named "test1_rep1_1.gz" & "test1_rep1_2.gz" for paired-end data and "test2_rep1_1.gz " & "test2_rep1_2.gz".
 - *'Flowcell_ID'*: unique identifier for a particular flow cell.
 - *'lane_ID'*: lane of the flow cell. If all samples used the same lane, set to "lane1" or any other name.
 - *'Technology'*: Sequencing technology used to generate the sequencing data. Valid values: ILLUMINA, SOLID, LS454, HELICOS and PACBIO. We only tested the pipeline for Illumina data so far.
 - *'Library_ID'*: library-specific identifier. If all samples used the same library, set to "default" or any other name.