<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 Biological motivation</td>
<td>47</td>
</tr>
<tr>
<td>12 Example dataset</td>
<td>49</td>
</tr>
<tr>
<td>13 Help, contribute and contact</td>
<td>51</td>
</tr>
<tr>
<td>14 Citation</td>
<td>53</td>
</tr>
<tr>
<td>15 Change log</td>
<td>55</td>
</tr>
<tr>
<td>16 License</td>
<td>59</td>
</tr>
</tbody>
</table>
Welcome to the *diffTF* documentation, and thank you for the interest in our software! These pages provide documentation and additional information for the *diffTF* pipeline.

To get yourself oriented, check the menu on the left or search what you are looking for in the search field in the upper left corner.

This site is organized into the following three parts:
**CHAPTER 1**

Try it out now!

---

`diffTF` runs on Linux and macOS and is even independent on the operating system if combined with `Singularity`. The following quick start briefly summarizes the necessary steps to install and use it.

Principally, there are two ways of installing `diffTF` and the proper tools:

1a. **The “easy” way:** Using `Singularity` and our preconfigured `diffTF` containers that contain all necessary tools, `R`, and `R` libraries

   You only need to install Snakemake (see below for details) and `Singularity`. Snakemake supports `Singularity` in Versions $\geq 2.4$. You can check whether you already have `Singularity` installed by simply typing

   ```bash
   singularity --version
   ```

   Snakemake requires at least version 2.4. If your version is below, please update to the latest `Singularity` version.

   **Note:** Make to read the section *Adaptations and notes when running with Singularity* properly!

1b. **The “more complicated” way:** Install the necessary tools (`Snakemake`, `samtools`, `bedtools`, `Subread`, and `R` along with various packages).

   **Note:** Note that all tools require Python 3.

   We recommend installing all tools except `R` via conda, in which case the installation then becomes as easy as

   ```bash
   conda config --add channels defaults
   conda config --add channels conda-forge
   conda config --add channels bioconda
   conda install snakemake bedtools samtools subread
   ```
If conda is not yet installed, follow the installation instructions. Installation is quick and easy. Make sure to open a new terminal after installation, so that conda is available.

**Note:** You do not need to uninstall other Python installations or packages in order to use conda. Even if you already have a system Python, another Python installation from a source such as the macOS Homebrew package manager and globally installed packages from pip such as pandas and NumPy, you do not need to uninstall, remove, or change any of them before using conda.

If you want to install the tools manually and outside of the conda framework, see the following instructions for each of the tools: snakemake, samtools, bedtools, Subread.

In addition, R is needed along with various packages (see below for details).

2. **Clone the Git repository:**

   ```bash
git clone https://git.embl.de/grp-zaugg/diffTF
   ``

   If you receive an error, Git may not be installed on your system. If you run Ubuntu, try the following command:

   ```bash
sudo apt-get install git
   ```

   For macOS, there are multiple ways of installing it. If you already have Homebrew (http://brew.sh) installed, simply type:

   ```bash
brew install git
   ```

   Otherwise, consult the internet on how to best install Git for your system.

3. **To run diffTF with an example ATAC-Seq / RNA-seq dataset for 50 TF, simply perform the following steps (see section Example dataset for dataset details):**

   - Change into the example/input directory within the Git repository
     ```bash
     cd diffTF/example/input
     ```

   - Download the data via the download script
     ```bash
     sh downloadAllData.sh
     ```

   - To test if the setup is correct, start a dryrun via the first helper script
     ```bash
     sh startAnalysisDryRun.sh
     ```

   - Once the dryrun is successful, start the analysis via the second helper script.
     ```bash
     sh startAnalysis.sh
     ```

   If you want to include Singularity (which we strongly recommend), simply edit the file and add the **--use-singularity** and **--singularity-args** command line arguments in addition to the other arguments (see the Snakemake documentation and the section Adaptations and notes when running with Singularity for more details).

   Thus, the command you execute should look like this:

   ```bash
   snakemake --snakefile ../../src/Snakefile --cores 2 --configfile config.json
            --use-singularity --singularity-args "--bind /your/diffTF/path"
   ```
Read in section *Adaptations and notes when running with Singularity* about the `--bind` option and what `/your/diffTF/path` means here, it is actually very easy!

You can also run the example analysis with all TF instead of only 50. For this, simply modify the `TF` parameter and set it to the special word `all` that tells *diffTF* to use all recognized TFs instead of a specific list only (see section *PARAMETER TFs* for details).

4. **To run your own analysis**, modify the files `config.json` and `sampleData.tsv`. See the instructions in the section *Run your own analysis* for more details.

5. **If your analysis finished successfully**, take a look into the `FINAL_OUTPUT` folder within your specified output directory, which contains the summary tables and visualization of your analysis. If you received an error, take a look in Section *Handling errors* to troubleshoot.
The only prerequisite here is that Snakemake and Singularity must be installed on the system you want to run \textit{diffTF}. See above for details with respect to the supported versions etc. For details how to install Snakemake, see below.
Prerequisites for the “manual” way

Note that most of this section is only relevant if you use Snakemake without Singularity. This section lists the required software and how to install them. As outlined in Section Try it out now!, the easiest way is to install all of them via conda. However, it is of course also possible to install the tools separately.

### 3.1 Snakemake

Please ensure that you have at least version 5.3 installed. Principally, there are multiple ways to install Snakemake. We recommend installing it, along with all the other required software, via conda.

### 3.2 samtools, bedtools, Subread

In addition, samtools, bedtools and Subread are needed to run diffTF. We recommend installing them, along with all the other required software, via conda.

### 3.3 R and R packages

A working R installation is needed and a number of packages from either CRAN or Bioconductor have to be installed. Type the following in R to install them:

```r
install.packages(c("checkmate", "futile.logger", "tidyverse", "reshape2", 
                   "RColorBrewer", "ggrepel", "lsr", "modeest", "boot", "grDevices", "pheatmap", 
                   "matrixStats", "locfdr", "pheatmap"))
source("https://bioconductor.org/biocLite.R")
biocLite(c("limma", "vsn", "csaw", "DESeq2", "DiffBind", "geneplotter", "Rsamtools"))
```
Run your own analysis

Running your own analysis is almost as easy as running the example analysis (see section Example dataset). Carefully read and follow the following steps and notes:

1. Copy the files config.json and startAnalysis.sh to a directory of your choice.

2. Modify the file config.json accordingly. For example, we strongly recommend running the analysis for all TF instead of just 50 as for the example analysis. For this, simply change the parameter “TFs” to “all”. See Section General configuration file for details about the meaning of the parameters. Do not delete or rename any parameters or sections.

3. Create a tab-separated file that defines the input data, in analogy to the file sampleData.tsv from the example analysis, and refer to that in the file config.json (parameter summaryFile).

4. Adapt the file startAnalysis.sh if necessary (the exact command line call to Snakemake and the various Snakemake-related parameters). If you run with Singularity, see the section below for modifications.

5. Since running the pipeline is often computationally demanding, read Section Executing diffTF - Running times and memory requirements and decide on which machine to run the pipeline. In most cases, we recommend running diffTF in a cluster environment (see Section Running diffTF in a cluster environment for details). The pipeline is written in Snakemake, and we strongly suggest to also read Section Working with diffTF and FAQs to get a basic understanding of how the pipeline works.
Adaptations and notes when running with Singularity

With Singularity, each rule will be executed in pre-configured isolated containers that contain all necessary tools. To enable it, you only have to add the following arguments when you execute Snakemake:

1. **--use-singularity**: Just type it like this, that’s all!

2. **--singularity-args**: You need to make all directories that contain files that are referenced in the `diffTF` configuration file available within the container also. By default, only the directory and subdirectories from which you start the analysis are automatically mounted inside the container. Since the `diffTF` source code is outside the input folder for the example analysis, however, at least the root directory of the Git repository has to be mounted. This is actually quite simple! Just use `--singularity-args --bind /your/diffTF/path` and replace `/your/diffTF/path` with the root path in which you cloned the `diffTF` Git repository (the one that has the subfolders `example`, `src` etc.). If you reference additional files, simply add one or multiple directories to the bind path (use the comma to separate them). For example, if you reference the files `/g/group1/user1/mm10.fa` and `/g/group2/user1/files/bla.txt` in the configuration file file, you may add `/g/group1/user1,/g/group2/user1/files` or even just `/g` to the bind path (as all files you reference are within `/g`).

3. **--singularity-prefix /your/directory** (optional): You do not have to, but you may want to add the `--singularity-prefix` argument to store all Singularity containers in a central place (here: `/your/directory`) instead of the local `.snakemake` directory. If you intend to run multiple `diffTF` analyses in different folders, you can save space and time because the containers won’t have to be downloaded each time and stored in multiple locations.

Please read the following additional notes and warnings related to Singularity:

- **Warning**: If you use Singularity version 3, make sure you have at least version 3.0.3 installed, as there was an issue with Snakemake and particular Singularity versions. For more details, see here.
We put the paper on bioRxiv, please read all methodological details here: Quantification of differential transcription factor activity and multiomic-based classification into activators and repressors: diffTF.

The workflow and conceptual idea behind diffTF is illustrated by the following three Figures. First, we give a high-level conceptual overview and a biological motivation:

![Fig. 1: Conceptual idea and workflow of diffTF](image)

Next, we show a schematic of the diffTF workflow from a more technical perspective by showing the actual steps that are performed:

We now show which rules are executed by Snakemake for a specific example (see the caption of the image):
Fig. 2: Schematic of the diffTF workflow, with input and output of the pipeline highlighted.
Fig. 3: Exact workflow (a so-called directed acyclic graph, or DAG) that is executed when calling Snakemake for an easy of example with two TFs (CEBPB and CTCF) for the two samples GMP.WT1 and MPP.WT1. Each node represents a rule name as defined in the Snakefile, and each arrow a dependency.
diffTF is implemented as a Snakemake pipeline. For a gentle introduction about Snakemake, see Section Working with diffTF and FAQs. As you can see, the workflow consists of the following steps or rules:

- **checkParameterValidity**: R script that checks whether the specified peak file has the correct format, whether the provided fasta file and the BAM files are compatible, and other checks
- **produceConsensusPeaks**: R script that generates the consensus peaks using the R package DiffBind if none are provided
- **filterSexChromosomesAndSortPeaks**: Filters various chromosomes (sex, unassembled ones, contigs, etc) from the peak file.
- **sortTFBSParallel**: Sort the TFBS lists by position
- **resortBAM**: Sort the BAM file for optimized processing (only run if data are paired-end)
- **intersectPeaksAndBAM**: Count all reads for peak regions across all input files
- **intersectTFBSAndBAM**: Intersect all TFBS with peak regions to retain only TFBS in peak regions
- **DiffPeaks**: R script that performs a differential accessibility analysis for the peak regions as well as sample permutations
- **analyzeTF**: R script that performs a TF-specific differential accessibility analysis
- **summary1**: R script that summarizes the previous script for all TFs
- **concatenateMotifs** and **concatenateMotifsPerm**: Concatenates previous results from either real or permuted data (TFBS motives)
- **calcNucleotideContent**: Calculates the GC content for all TFBS
- **binningTF**: R script that performs the binning approach in a TF-specific manner
- **summaryFinal**: R script that summarizes the analysis and calculates final statistics
- **cleanUpLogFiles**: Cleans up the LOGS_AND_BENCHMARKS directory (mostly relevant if run in cluster mode)
7.1 Summary

As input for diffTF for your own analysis, the following data are needed:

- **BAM** file with aligned reads for each sample (see `PARAMETER summaryFile`)
- genome reference *fasta* that has been used to produce the **BAM** files (see `PARAMETER refGenome_fasta`)
- Optionally: corresponding RNA-Seq data (see `PARAMETER RNASeqCounts`)

In addition, the following files are needed, all of which we provide already for human hg19, hg38 and mouse mm10:

- TF-specific list of TFBS (see `PARAMETER dir_TFBS`)
- mapping table (see `PARAMETER HOCOMOCO_mapping`)

Lastly, some metadata files are needed that specify diffTF-specific and Snakemake-specific parameters. They are explained in detail in the next sections. If this sounds complicated, don’t worry, just take the example analysis, and you will understand within a few minutes what these files are:

- a general configuration file (*General configuration file*)
- a metadata file for the samples (*Input metadata*)

 optionally, if run on a cluster, a cluster configuration file (see in particular the Snakemake documentation for details, but we also provide example cluster files as well as Section *Running diffTF in a cluster environment*)

7.2 General configuration file

To run the pipeline, a configuration file that defines various parameters of the pipeline is required.

**Note:** Please note the following important points:

- the name of this file is irrelevant, but it must be in the right format (JSON) and it must be referenced correctly when calling *Snakemake* (via the --configfile parameter). We recommend naming it *config.json*
In the following, we explain all parameters in detail, organized by section names.

### 7.2.1 SECTION **par_general**

**PARAMETER outdir**

**Summary** String. Default “output”. Root output directory.

**Details** The root output directory where all output is stored.

**PARAMETER maxCoresPerRule**

**Summary** Integer > 0. Default 16. Maximum number of cores to use for rules that support multithreading.

**Details** This affects currently only rules involving `featureCounts` - that is, `intersectPeaksAndBAM` while for rule `intersectTFBSAndBAM`, the number of cores is hard-coded to 4. When running `Snakemake` locally, each rule will use at most this number of cores, while in a cluster setting, this value refers to the maximum number of CPUs an individual job / rule will occupy. If the node the job is executed on has fewer nodes, then the maximum number of cores on the node will be taken.

**PARAMETER regionExtension**

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

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

**PARAMETER comparisonType**

**Summary** String. Default “”.

**Details** 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.

**PARAMETER conditionComparison**

**Summary** String. Default “”. Specifies the two conditions you want to compare.

**Details** This parameter specifies the contrast you are making in `diffTF`. Two conditions have to be specified, separated by a comma. For example, if you want to compare GMP and MPP samples, the parameter should be “GMP,MPP”. Both conditions have to be present in the column “conditionSummary” in the sample file table (see parameter `summaryFile` (PARAMETER `summaryFile`)).
Note: The order of the two conditions matters. The condition specified first is the reference condition. For the “GMP,MPP” example, all log2 fold-changes will be the log2fc of MPP as compared to GMP. That means that a positive log2 fold-change means it is higher in MPP as compared to GMP. This is particularly relevant for the allMotifs output file.

PARAMETER designContrast


Details This important parameter defines the actual contrast that is done in the differential analysis. That is, which groups of samples are being compared? Examples include mutant vs wild type, mutated vs. unmutated, etc. The last element in the formula must always be conditionSummary, which defines the two groups that are being compared. This name is currently hard-coded and required by the pipeline. Our pipeline allows including additional variables to model potential confounding variables, like gender, batches etc. For each additional variable that is part of the formula, a corresponding and identically named column in the sample summary file must be specified. For example, for an analysis that also includes the batch number of the samples, you may specify this as “~ Treatment + conditionSummary”.

PARAMETER designVariableTypes

Summary String. Default conditionSummary:factor. The data types of all elements listed in designContrast (PARAMETER designContrast).

Details 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”. For example, if designContrast (PARAMETER designContrast) is specified as “~ Treatment + conditionSummary”, the corresponding types might be “Treatment:factor, conditionSummary:factor”. If a data type is specified as either “logical” or “factor”, the variable will be treated as a discrete variable with a finite number of distinct possibilities (something like batch, for example). conditionSummary is usually specified as factor because you want to make a pairwise comparison of exactly two conditions. If conditionSummary is specified as “integer” or “numeric”, however, the variable is treated as continuously-scaled, which changes the interpretation of the results, see the note below.

Note: Importantly, if the variable of interest is continuous-valued (i.e., marked as being integer or numeric), then the reported log2 fold change is per unit of change of that variable. That is, in the final Volcano plot, TFs displayed in the left side have a negative slope per unit of change of that variable, while TFs at the right side have a positive one.

PARAMETER nPermutations

Summary Integer >= 0. Default 50. The number of random sample permutations.

Details If set to a value > 0, in addition to the real data, the sample conditions as specified in the sample table will be randomly permuted nPermutations times. This is the recommended way of computing statistical significances for each TF. In this approach, the resulting significance value captures the significance of the effect size (that is, the TF activity) for the real data as compared to permuted one. Note that the maximum number of possible permutations is limited by the number of samples and can be computed with the binomial coefficient n over k. For example, if you have n = 8 samples in total and they split up in the two conditions/groups as k = 5 / k = 3, the total number of permutations is 8 over 5 or 8 over 3 (they are both identical). We generally recommend setting this value to high values such as 1,000. If the value is set to a number higher than the number of possible
permutations, it will be adjusted automatically to the maximum number of permutations as determined by the binomial coefficient.

If set to 0, an alternative way of computing significances that is not based on permutations is performed. First, in the CG normalization step, a Welch Two Sample t-test is performed for each bin and the overall significance by treating the T-statistics as z-scores is calculated, which allows to summarize them across the bins and convert them to one p-value per TF. For this conversion of z-scores per bin to p-value an estimate of the variance of the T-scores is approximated (see the publication for details). This procedure reduces the dependency of the p-value on the sample size (since the number of TFBS can range between a few dozen and multiple tens of thousands depending on the TF).

**Note:** If set to a value > 0, the parameter \( n_{\text{Bootstraps}} \) \((\text{PARAMETER} \ n_{\text{Bootstraps}})\) is ignored and can be set to any value.

**Note:** While using permutations is the recommended approach for assessing statistical significance, in some cases it might be useful to use the alternative approach: If the number of samples is small or the groups show a very uneven distributions, the number of possible permutations is very small and therefore also the permutation-based approach might not accurately assess significance.

**Note:** The permutation-based approach is computationally more expensive than the analytical approach. The running time of the pipeline increases with the number of permutations.

**Warning:** Do not change the value of this parameter after (parts of) the pipeline have been run, some steps may fail due to this change. If you really need to change the value, rerun the pipeline from the \textit{diffPeaks} step onwards.

**PARAMETER** \( n_{\text{Bootstraps}} \)

**Summary** Integer \( \geq 0 \). Default 1,000. The number of bootstrap for estimating the variance of the TF-specific T scores in the CG binning step.

**Details** To properly estimate the variance of the T scores for each TF in the CG binning step, we employ a bootstrap approach using the boot library in R with a user-adjustable number of bootstrap replicates (default 1,000), with resampling the bin-specific data and then performing the t-test against the full sample as described above. We then calculate the variance of the bootstrapped T scores for each bin. For more details, see the methods of the publication.

**Note:** Only relevant if the parameter \( n_{\text{Permutations}} \) \((\text{PARAMETER} \ n_{\text{Permutations}})\) is set to 0. If both are set to 0, an error is thrown.

**Warning:** If bootstraps are used, it is recommended to use a reasonable large number. We recommend a value 1,000 and found that higher numbers do not add much benefit but instead only increase running time unnecessarily.
PARAMETER nGCBins

Summary  Integer > 0. Default 10. Number of GC bins for the binning step.

Details  This parameter sets the number of GC bins that are used during the binning step. The default is to split the data into 10 bins (0-10% GC content, 11-20%, …, 91-100%), for each of which the significance is calculated independently (see Methods). Too many bins may result in bins being skipped due to an insufficient number of TFBS for that particular bin and TF, while too few bins may introduce GC-specific biases when summarizing the signal across all TFBS.

PARAMETER TFs

Summary  String. Default “all”. Either “all” or a comma-separated list of TF names of TFs to include. If set to “all”, all TFs that are found in the directory as specified in dir_TFBS (PARAMETER dir_TFBS) will be used.

Details  If the analysis should be restricted to a subset of TFs, list the names of the TF to include in a comma-separated manner here.

Note:  For each TF {TF}, a corresponding file {TF}_TFBS.bed needs to be present in the directory that is specified by dir_TFBS (PARAMETER dir_TFBS).

Warning:  We strongly recommending running diffTF with as many TF as possible due to our statistical model that we use that compares against a background model.

PARAMETER dir_scripts

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

Details  

Warning:  The folder name must be R, and it has to be located in the same folder as the Snakefile.

PARAMETER RNASeqIntegration

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

Details  If set to true, RNA-Seq counts as specified in RNASeqCounts (PARAMETER RNASeqCounts) will be used to classify each TF into either “activator”, “repressor”, “unknown”, or “not-expressed” for the final Volcano plot visualization and the summary table.

7.2.2 SECTION samples

PARAMETER summaryFile

Summary  String. Default “samples.tsv”. Path to the sample metadata file.

Details  Path to a tab-separated file that summarizes the input data. See the section Input metadata and the example file for how this file should look like.
PARAMETER pairedEnd

Summary Logical. true or false. Default true. Is the data paired-end? If single-end, set to false.
Details Both paired-end and single-end data can be run with diffTF.

7.2.3 SECTION peaks

PARAMETER consensusPeaks

Details If set to the empty string “”, the pipeline will generate a consensus peaks out of the peak files from each individual sample using the R package DiffBind. 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 section Input metadata for details.
- The format of the peak files, as specified in peakType (PARAMETER peakType)
If a file is provided, it must be a valid BED file with at least 3 columns:
- tab-separated columns
- no column names in the first row
- Columns 1 to 3:
  1. Chromosome
  2. Start position
  3. End position
- Optional (content for each is ignored and not checked for validity):
  4. Identifier (will be made unique for each if this is not the case already)
  5. Score
  6. Strand

Warning: diffTF will take a long time to run if the number of peaks is too high. We recommend having less than 100,000 peaks. If the number of peaks is higher for your analysis, we strongly recommend filtering the peaks beforehand to include only the most relevant peaks.

PARAMETER peakType

Summary String. Default narrow. File format of the individual, sample-specific peak files. Only relevant if no consensus peak file has been provided (i.e., the PARAMETER consensusPeaks is empty).
Details Only needed if no consensus peak set has been provided. All individual peak files must be in the same format. We recommend the narrow format (files ending in .narrowPeak) that is a direct output from MACS2, but other formats are supported. See the help for DiffBind dba for a full list of supported formats, the most common ones include:
- bed: .bed file; peak score is in fifth column
- narrow: narrowPeaks file (from MACS2)
PARAMETER `minOverlap`

**Summary** 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 relevant if no consensus peak file has been provided (i.e., `consensusPeaks` is empty).

**Details** Only include peaks in at least this many peak sets in the main binding matrix. If set to a value between zero and one, peak will be included from at least this proportion of peak sets. For more information, see the `minOverlap` argument in the `dba` function of `DiffBind` (see here).

### 7.2.4 SECTION additionalInputFiles

PARAMETER `refGenome_fasta`


**Details**

**Warning:** You need write access to the directory in which the `fasta` file is stored, make sure this is the case or copy the `fasta` file to a different directory. The reason is that the pipeline produces a `fasta` index file, which is put in the same directory as the corresponding `fasta` file. This is a limitation of `samtools faidx` and not our pipeline.

**Note:** This file has to be in concordance with the input data; that is, the exact same genome assembly version must be used. In the first step of the pipeline, this is checked explicitly, and any mismatches will result in an error.

PARAMETER `dir_TFBS`

**Summary** String. Path to the directory where the TF-specific files for TFBS results are stored.

**Details** Each TF `{TF}` has to have one `BED` file, in the format `{TF}.bed`. Each file must be a valid `BED6` file with 6 columns, as follows:

1. chromosome
2. start
3. end
4. ID (or sequence)
5. score or any other numeric column
6. strand

For user convenience, we provide such sorted files as described in the publication as a separate download:

- `hg19`: For a pre-compiled list of 638 human TF with in-silico predicted TFBS based on the `HOCOMOCO 10` database and `PWMScan` for `hg19`, download this file:

- `hg38`: For a pre-compiled list of 767 human TF with in-silico predicted TFBS based on the `HOCOMOCO 11` database and `FIMO` from the MEME suite for `hg38`, download this file. For a pre-compiled list of 768
human TF with in-silico predicted TFBS based on the HOCOMOCO 11 database and PWMScan for hg38, download this file:

- mm10: For a pre-compiled list of 422 mouse TF with in-silico predicted TFBS based on the HOCOMOCO 10 database and PWMScan for mm10, download this file:

However, you may also manually create these files to include additional TF of your choice or to be more or less stringent with the predicted TFBS. For this, you only need PWMs for the TF of interest and then a motif prediction tool such as FIMO or MOODS.

**PARAMETER RNASeqCounts**

**Summary** String. Default “”. Path to the file with RNA-Seq counts.

**Details** If no RNA-Seq data is included, set to the empty string “”. Otherwise, if RNASeqIntegration (**PARAMETER RNASeqIntegration**) is set to true, specify the path to a tab-separated file with normalized RNA-Seq counts. It does not matter whether the values have been variance-stabilized or not, as long as values across samples are comparable. Also, consider filtering lowly expressed genes. For guidance, you may want to read Question 4 here.

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 (**summaryFile**, **PARAMETER summaryFile**). If you have RNA-Seq data for only a subset of the input samples, this is no problem - the classification will then naturally only be based on the subset. 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 from the file as specified in HOCOMOCO_mapping (**PARAMETER HOCOMOCO_mapping**).

**PARAMETER HOCOMOCO_mapping**

**Summary** String. Path to the TF-Gene translation table.

**Details** 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 compatible with the pre-provided TFBS lists. Specifically, for each of the currently three TFBS lists, we provide corresponding translation tables for:

1. hg19 with HOCOMOCO 10
2. hg38 with HOCOMOCO 11
3. mm10 with HOCOMOCO 10

If you want to create your own version, check the example translation tables and construct one with an identical structure.

### 7.3 Input metadata

This file summarizes the data and corresponding available metadata that should be used for the analysis. The format is flexible and may contain additional columns that are ignored by the pipeline, so it can be used to capture all 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.

**Warning:** Make sure that the line endings are correct. Different operating systems use different characters to mark the end of line, and the line ending character must be compatible with the operation system in which you run diffTF. For example, if you created the file in MAC, but you run it in a Linux...
environment (e.g., a cluster system), you may have to convert line endings to make them compatible with Linux. For more information, see here.

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.

**Warning:** All BAM files must meet SAM format specifications. You may use the program ValidateSamFile from the Picard tools to check and identify problems with your file. Chromosome names must have a “chr” as prefix, otherwise diffTF may crash.

- **peaks**: absolute path to the sample-specific peak file, in the format as given by `peakType` (PARAMETER `peakType`). Only needed if no consensus peak file is provided.
- **conditionSummary**: String with an arbitrary condition name that defines which condition the sample belongs to. There must be only exactly two different conditions across all samples (e.g., mutated and unmutated, day0 and day10, ...). In addition, the two conditions must match the ones specified in the parameter `conditionComparison` (PARAMETER `conditionComparison`).
- if applicable, all additional variables from the design formula except `conditionSummary` must also be present as a separate column.

**Warning:** Do not change the samples data after you started an analysis. You may introduce inconsistencies that will result in error messages. If you need to alter the sample data, we strongly advise to recalculate all steps in the pipeline.
The pipeline produces quite a large number of output files, only some of which are however relevant for the regular user.

**Note:** In the following, the directory structure and the files are briefly outlined. As some directory or file names depend on specific parameters in the configuration file, curly brackets will be used to denote that the filename depends on a particular parameter or name. For example, \{comparisonType\} and \{regionExtension\} refer to `comparisonType` (**PARAMETER comparisonType**) and `regionExtension` (**PARAMETER regionExtension**) as specified in the configuration file.

Most files have one of the following file formats:

- .bed.gz (gzipped bed file)
- .tsv.gz (tab-separated value, text file with tab as column separators, gzipped)
- .rds (binary R format, read into with the function `readRDS`)
- .pdf (PDF format)
- .log (text format)

### 8.1 FOLDER FINAL_OUTPUT

In this folder, the final output files are stored. Most users want to examine the files in here for further analysis.

#### 8.1.1 Sub-folder extension{regionExtension}

Stores results related to the user-specified extension size (regionExtension, **PARAMETER regionExtension**)
Note: In all output files, in the column permutation, 0 always refers to the non-permuted, real data, while permutations > 0 reflect real permutations.

**FILE {comparisonType}.allMotifs.tsv.gz**

**Summary** Summary table for each TFBS

**Details** Columns are as follows:

- **permutation**: The number of the permutation. This will always be 0, so it can be ignored essentially in this file.
- **TF**: name of the TF
- **chr, MSS, MES, strand, TFBSID**: Genomic location and identifier of the (extended) TFBS
- **peakID**: Genomic location and annotation of the overlapping peak region
- **l2FC, pval, pval_adj**: Results from the limma or DESeq2 analysis, see the respective documentation for details (see below for links and further explanation). These column names are shared between limma and DESeq2. l2FC are interpreted as described in the parameter conditionComparison (PARAMETER conditionComparison)
- **DESeq_baseMean, DESeq_ldcSE, DESeq_stat**: Results from the DESeq2 analysis, see the DESeq2 documentation for details (e.g., ?DESeq2::results). If DESeq2 was not run for calculating log2 fold-changes (i.e., if the value for the parameter nPermutations (PARAMETER regionExtension) is >0), these columns are set to NA.
- **limma_avgExpr, limma_B, limma_t_stat**: Results from the limma analysis, see the limma documentation for details (e.g., ?topTable). If limma was not run (i.e., if the value for the parameter nPermutations (PARAMETER regionExtension) is 0), these columns are set to NA.

**FILE {comparisonType}.TF_vs_peak_distribution.tsv.gz**

**Summary** This summary table contains various results regarding TFs, their log2 fold change distribution across all TFBS and differences between all TFBS and the peaks

**Details** See the description of the file {TF}.{comparisonType}.summary.rds. This file aggregates the data for all TF and adds the following additional columns: - pvalue_adj: adjusted (fdr aka BH) p-value (based on pvalue_raw) - Diff_mean, Diff_median, Diff_mode, Diff_skew: Difference of the mean, median, mode, and skewness between the log2 fold-change distribution across all TFBS and the peaks, respectively

**FILE {comparisonType}.summary.tsv.gz**

**Summary** The final summary table that is also used for the final Volcano plot visualization.

**Details** The columns are as follows:

- **TF**: name of the TF
- **weighted_meanDifference**: the weighted mean difference of the real and background distribution across all CG bins. This value is the basis for the final calculation of the x-axis position for the Volcano plot.
- **TFBS**: The number of TF binding sites for the particular TF that overlap with the peaks
- **fdr**: the local FDR value that is derived from comparing the observed values against the permuted ones
• classification: RNA-Seq classification (either activator, undetermined, repressor or not-expressed)

**FILE** `{comparisonType}.diagnosticPlots.pdf`

**Summary** Various diagnostic plots for the final TF activity values. TODO UPDATE

**Details** If the number of permutations is larger than 0, the first three pages show various versions of the permuted weighted_meanDifference values and how they relate to the real ones. Permutation 0, as used everywhere throughout the pipeline, contains the real values, while any permutation > 0 refers to an actual permutation. Page 1 shows real and permuted values, page 2 only permuted ones, page 3 a density plot of the real values with the permutation thresholds as dashed lines, inside of which TFs are not labeled as they fall within the permutation and therefore noise area. The next page shows various diagnostic plots from the *locfdr* package to estimate the distribution median, while the remaining plots show histograms of all relevant columns in the final output table for different sets of TFs depending on a specific FDR threshold.

### 8.2 FOLDER PEAKS

Stores peak-associated files.

#### 8.2.1 FILES `{comparisonType}.consensusPeaks.filtered.sorted.bed`

**Summary** Only present if no consensus peak file was provided (consensusPeaks, *PARAMETER* consensusPeaks). Produced in rule `filterSexChromosomesAndSortPeaks`. Generated consensus peaks, before filtering (see below).

**Details** Filtered consensus peaks (removal of peaks from one of the following chromosomes: chrX, chrY, chrM, chrUn*, and all contig names that do not start with “chr” such as *random* or *hapl*_gl*

#### 8.2.2 FILE `{comparisonType}.allBams.peaks.overlaps.bed.gz`

**Summary** Produced in rule `intersectPeaksAndBAM`. Counts for each consensus peak with each of the input BAM files.

**Details** No details provided yet.

#### 8.2.3 FILE `{comparisonType}.sampleMetadata.rds`

**Summary** Produced in rule `DiffPeaks`. Stores data for the input data (similar to the input sample table), for both the real data and the permutations.

**Details** No details provided yet.

#### 8.2.4 FILE `{comparisonType}.peaks.rds`

**Summary** Produced in rule `DiffPeaks`. Stores all peaks that will be used in the analysis.

**Details** No details provided yet.
8.2.5 FILE \{comparisonType\}.peaks.tsv.gz

**Summary**  Produced in rule `DiffPeaks`. Stores the results of the differential accessibility analysis for the peaks.

**Details**  No details provided yet.

8.2.6 FILE \{comparisonType\}.normFacs.rds

**Summary**  Produced in rule `DiffPeaks`. Gene-specific normalization factors for each sample and peak.

**Details**  This file is produces after the differential accessibility analysis for the peaks. The normalization factors are used for the TF-specific differential accessibility analysis.

8.2.7 FILES \{comparisonType\}.diagnosticPlots.peaks.pdf

**Summary**  Produced in rule `DiffPeaks`. Various diagnostic plots for the differential accessibility peak analysis for the real data

**Details**  The pages are as follows:

1. MA plots
2. density plots of normalized and non-normalized counts
3. mean-average plots (average of the log-transformed counts vs the fold-change per peak) for each of the sample pairs
4. mean SD plots (row standard deviations versus row means)

8.2.8 FILE \{comparisonType\}.DESeq.object.rds

**Summary**  Produced in rule `DiffPeaks`. The `DESeq2` object from the differential accessibility peak analysis.

**Details**  If the number of permutations (parameter `nPermutations` \((PARAMETER nPermutations)\) is set to 0, `DESeq` is fully run, otherwise the objects does only contain the counts and metadata but no `results` slot.

8.3 FOLDER TF–SPECIFIC

Stores TF-specific files. For each TF \(\{\text{TF}\}\), a separate sub-folder \(\{\text{TF}\}\) is created by the pipeline. Within this folder, the following structure is created:

8.3.1 Sub-folder `extension\{regionExtension\}`

**FILES**  \(\{\text{TF}\}\).\{comparisonType\}.allBAMs.overlaps.bed.gz and \(\{\text{TF}\}\).\{comparisonType\}.allBAMs.overlaps.bed.summary

**Summary**  Overlap and `featureCounts` summary file of read counts across all TFBS for all input `BAM` files.

**Details**  For more details, see the documentation of `featureCounts`. 
**FILE** `{TF}.{comparisonType}.output.tsv.gz`  
**Summary** Produced in rule `analyzeTF`. A summary table for the *limma* analysis.  
**Details** See the file `{comparisonType}.allMotifs.tsv.gz` in the `FINAL_OUTPUT` folder for a column description.

**FILE** `{TF}.{comparisonType}.outputPerm.tsv.gz`  
**Summary** Produced in rule `analyzeTF`. A subset of the file `{TF}.{comparisonType}.output.tsv.gz` that stores only the necessary permutation-specific results for subsequent steps.  
**Details** This file has the following columns (see the description for the file `{TF}.{comparisonType}.output.tsv.gz` for details): - *TF* - *TFBSID* - *log2fc_perm* columns, which store the permutation-specific log2 fold-changes of the particular TFBS. Permutation 0 refers to the real data.

**FILE** `{TF}.{comparisonType}.summary.rds`  
**Summary** Produced in rule `analyzeTF`. A summary table for the log2 fold-changes across all TFBS *limma* results.  
**Details** This file summarizes the TF-specific results for the differential analysis and has the following columns: - *TF*: name of the TF - *permutation*: The number of the permutation. - *Pos_l2FC*, *Mean_l2FC*, *Median_l2FC*, *sd_l2FC*, *Mode_l2FC*, *skewness_l2FC*: fraction of positive values, mean, median, standard deviation, mode value and Bickel’s measure of skewness of the log2 fold change distribution across all TFBS - *pvalue_raw* and *pvalue_adj*: raw and adjusted (fdr aka BH) p-value of the t-test - *T_statistic*: the value of the T statistic from the t-test - *TFBS_num*: number of TFBS

**FILES** `{TF}.{comparisonType}.diagnosticPlots.pdf`  
**Summary** Produced in rule `analyzeTF`. Various diagnostic plots for the differential accessibility TFBS analysis for the real data.  
**Details** See the description of the file `{comparisonType}.diagnosticPlots.peaks.pdf` in the `PEAKS` folder, which has an identical structure. Here, the second last page shows a density plot of the log2 fold-changes for the specific pairwise condition that the user selected, separately for the peaks only and across all TFBS from the specific TF. The last page shows the same but in a cumulative representation.

**FILE** `{TF}.{comparisonType}.permutationResults.rds`  
**Summary** Produced in rule `binningTF`. Contains a data frame that stores the results of bin-specific results.  
**Details** No details provided yet.

**FILE** `{TF}.{comparisonType}.permutationSummary.tsv.gz`  
**Summary** Produced in rule `binningTF`. A final summary table that summarizes the results across bins by calculating weighted means.  
**Details** The data of this table are used for the final visualization.
**FILE** `{TF}.({comparisonType}).covarianceResults.rds`

**Summary**  Produced in rule `binningTF`. Contains a data frame that stores the results of the pairwise bin covariances and the bin-specific weights.

**Details**

---

**Note:** Covariances are only computed for the real data but not the permuted ones.

### 8.4 FOLDER LOGS_AND_BENCHMARKS

Stores various log and error files.

- ***.log files from R scripts:** Each log file is produced by the corresponding R script and contains debugging information as well as warnings and errors:
  - `checkParameterValidity.R.log`
  - `produceConsensusPeaks.R.log`
  - `diffPeaks.R.log`
  - `analyzeTF.{TF}.R.log` for each TF `{TF}`
  - `summary1.R.log`
  - `binningTF.{TF}.log` for each TF `{TF}`
  - `summaryFinal.R.log`

- ***.log summary files:** Summary logs for user convenience, produced at very end of the pipeline only. They should contain all errors and warnings from the pipeline run.
  - `all.errors.log`
  - `all.warnings.log`

### 8.5 FOLDER TEMP

Stores temporary and intermediate files. Since they are usually not relevant for the user, they are explained only very briefly here.

#### 8.5.1 Sub-folder SortedBAM

Stores sorted versions of the original BAMs that are optimized for fast count retrieval using `featureCounts`. Only present if data are paired-end.

- `{basenameBAM}.bam` for each input BAM file: Produced in rule `resortBAM`. Resorted BAM file

#### 8.5.2 Sub-folder extension{regionExtension}

Stores results related to the user-specified extension size `{regionExtension, PARAMETER regionExtension}`
• `{comparisonType}.allTFBS.peaks.bed.gz`: Produced in rule `intersectPeaksAndTFBS`. *BED* file containing all TFBS from all TF that overlap with the peaks after motif extension.

• `conditionComparison.rds`: Produced in rule `DiffPeaks`. Stores the condition comparison as a string. Some steps in diffTF need this file as input.

• `{comparisonType}.motifs.coord.permutation{perm}.bed.gz` and `{comparisonType}.motifs.coord.nucContent.permutation{perm}.bed.gz` for each permutation `{perm}`: Produced in rule `calcNucleotideContent`, and needed subsequently for the binning. Temporary and result file of `bedtools nuc`, respectively. The latter contains the GC content for all TFBS.

• `{comparisonType}.checkParameterValidity.done`: temporary flag file

• `{TF}.TFBS.sorted.bed` for each TF `{TF}`: Produced in rule `sortTFBSParallel`. Coordinate-sorted version of the input TFBS. Only “regular” chromosomes starting with “chr” are kept, while sex chromosomes (chrX, chrY), chrM and unassembled contigs such as chrUn are additionally removed.

• `{comparisonType}.allTFBS.peaks.bed.gz`: Produced in rule `intersectPeaksAndTFBS`. *BED* file containing all TFBS from all TF that overlap with the peaks before motif extension.
CHAPTER 9

Working with \textit{diffTF} and FAQs

9.1 General remarks

diffTF is programmed as a \textit{Snakemake} pipeline. \textit{Snakemake} is a bioinformatics workflow manager that uses workflows that are described via a human readable, Python based language. It offers many advantages to the user because each step can easily be modified, parts of the pipeline can be rerun, and workflows can be seamlessly scaled to server, cluster, grid and cloud environments, without the need to modify the workflow definition or only minimal modifications. However, with great flexibility comes a price: the learning curve to work with the pipeline might be a bit higher, especially if you have no \textit{Snakemake} experience. For a deeper understanding and troubleshooting errors, some knowledge of \textit{Snakemake} is invaluable.

Simply put, \textit{Snakemake} executes various rules. Each rule can be thought of as a single recipe or task such as sorting a file, running an R script, etc. Each rule has, among other features, a name, an input, an output, and the command that is executed. You can see in the \textit{Snakefile} what these rules are and what they do. During the execution, the rule name is displayed, so you know exactly at which step the pipeline is at the given moment. Different rules are connected through their input and output files, so that the output of one rule becomes the input for a subsequent rule, thereby creating dependencies, which ultimately leads to the directed acyclic graph (\textit{DAG}) that describes the whole workflow. You have seen such a graph in Section \textit{Workflow}.

In diffTF, a rule is typically executed separately for each TF. One example for a particular rule is sorting the TFBS list for the TF CTCF.

In diffTF, the total number of \textit{jobs} or rules to execute can roughly be approximated as $3 \times n_{TF}$, where $n_{TF}$ stands for the number of TFs that are included in the analysis. For each TF, three sets of rules are executed:

1. Calculating read counts for each TFBS within the peak regions (rule \texttt{intersectTFBSAndBAM})
2. Differential accessibility analysis (rule \texttt{analyzeTF})
3. Binning step (rule \texttt{binningTF})

In addition, one rule per permutation is executed, so an additional $n_{Permutations}$ rules are performed. Lastly, a few other rules are executed that however do not add up much more to the overall rule count.
9.2 Executing diffTF - Running times and memory requirements

`diffTF` can be computationally demanding depending on the sample size and the number of peaks. In the following, we discuss various issues related to time and memory requirements and we provide some general guidelines that worked well for us.

**Warning:** We generally advise to run `diffTF` in a cluster environment. For small analysis, a local analysis on your machine might work just fine (see the example analysis in the Git repository), but running time increases substantially due to limited amount of available cores.

9.2.1 Analysis size

We now provide a *very rough* classification into small, medium and large with respect to the sample size and the number of peaks:

- **Small:** Fewer than 10-15 samples, number of peaks not exceeding 50,000-80,000, normal read depth per sample
- **Large:** Number of samples larger than say 20 or number of peaks clearly exceeds 100,000, or very high read depth per sample
- **Medium:** Anything between small and large

9.2.2 Memory

Some notes regarding memory:

- **Disk space:** Make sure you have enough space left on your device. As a guideline, analysis with 8 samples need around 12 GB of disk space, while a large analysis with 84 samples needs around 45 GB. The number of permutations also has an influence on the (temporary) required storage and a high number of permutations (> 500) may substantially increase the memory footprint. Note that most space is occupied in the `TEMP` folder, which can be deleted after an analysis has been run successfully. We note, however, that rerunning (parts of) the analysis will require regenerating files from the `TEMP` folder, so only delete the folder or files if you are sure that you do not need them anymore.
- **Machine memory:** Although most steps of the pipeline have a modest memory footprint of less than 4 GB or so, depending on the analysis size, some may need 10+ GB of RAM during execution. We therefore recommend having at least 10 GB available for large analysis (see above).

9.2.3 Number of cores

Some notes regarding the number of available cores:

- `diffTF` can be invoked in a highly parallelized manner, so the more CPUs are available, the better.
- you can use the `--cores` option when invoking `Snakemake` to specify the number of cores that are available for the analysis. If you specify 4 cores, for example, up to 4 rules can be run in parallel (if each of them occupies only 1 core), or 1 rule can use up to 4 cores.
- we strongly recommend running `diffTF` in a cluster environment due to the massive parallelization. With `Snakemake`, it is easy to run `diffTF` in a cluster setting. Simply do the following:
  - write a cluster configuration file that specifies which resources each rule needs. For guidance and user convenience, we provide different cluster configuration files for a small and large analysis. See the folder

Chapter 9. Working with `diffTF` and FAQs
9.2.4 Total running time

Some notes regarding the total running time:

- the total running time is very difficult to estimate beforehand and depends on many parameters, most importantly the number of samples, their read depth, the number of peaks, and the number of TF included in the analysis.
- for small analysis such as the example analysis in the Git repository, running times are roughly 30 minutes with 2 cores for 50 TF and a few hours with all 640 TF.
- for large analysis, running time will be up to a day or so when executed on a cluster machine

9.3 Running diffTF in a cluster environment

If diffTF should be run in a cluster environment, the changes are minimal due to the flexibility of Snakemake. You only need to change the following:

- create a cluster configuration file in JSON format. See the files in the clusterConfigurationTemplates folder for examples. In a nutshell, this file specifies the computational requirements and job details for each job that is run via Snakemake.
- invoke Snakemake with a cluster parameter. As an example, you may use the following for a SLURM cluster:

```
snakemake -s path/to/Snakefile
   --configfile path/to/configfile --latency-wait 30
   --notemp --rerun-incomplete --reason --keep-going
   --cores 16 --local-cores 1 --jobs 400
   --cluster-config path/to/clusterconfigfile
   --cluster " sbatch -p {cluster.queue} -J {cluster.name} \
       --cpus-per-task {cluster.nCPUs} \ 
       --mem {cluster.memory} --time {cluster.maxTime} -o "{cluster.output}" \ 
       -e "{cluster.error}" --mail-type=None --parsable "
```

- the corresponding cluster configuration file might look like this:

```
{
    "__default__": {
        "queue": "htc",
        "nCPUs": "{threads}",
        "memory": 2000,
        "maxTime": "1:00:00",
        "name": "{rule}.{wildcards}",
        "output": "{rule}.{wildcards}.out",
        "error": "{rule}.{wildcards}.err"
    },
    "resortBAM": {
        "memory": 5000,
        "maxTime": "1:00:00"
    }
}
```

(continues on next page)
"intersectPeaksAndPWM": {
    "memory": 5000,
    "maxTime": "1:00:00"
},
"intersectPeaksAndBAM": {
    "memory": 5000,
    "maxTime": "1:00:00"
},
"intersectTFBSAndBAM": {
    "memory": 5000,
    "maxTime": "1:00:00"
},
"DiffPeaks": {
    "memory": 5000,
    "maxTime": "1:00:00"
},
"analyzeTF": {
    "memory": 5000,
    "maxTime": "1:00:00"
},
"binningTF": {
    "memory": 5000,
    "maxTime": "1:00:00"
},
"summaryFinal": {
    "memory": 5000,
    "maxTime": "0:30:00"
},
"cleanUpLogFiles": {
    "memory": 1000,
    "maxTime": "0:30:00"
}

A few motes might help you to get started:

• each name in the --cluster argument string from the command line (here: queue, name nCPUs, memory, maxTime, output, and error) must appear also in the __default__ section of the referenced cluster configuration file (via --cluster-config)

• for brevity here, only rules with requirements different from the specified default have been included here in the online version, while the templates in the repository contain all rules, even if they have the same requirements as the default. The latter makes it easier for practical purposes to change requirements later on

• the --cluster argument is the only part that has to be adjusted for your cluster system. It is quite simple really, you essentially just link the content of the configuration file to the cluster system you want to submit the jobs to. More specifically, you refer to the cluster configuration file via the cluster. string, followed by the name of the parameter in the cluster configuration. For parameters that refer to filenames, an extra escaped quotation mark \" has been added so that the command also works in case of spaces in filenames (which should always be avoided at all costs)

• the cluster configuration file has multiple sections defined that correspond to the names of the rules as defined in the Snakefile, plus the special section __default__ at the very top, the latter of which specifies the default cluster options that apply to all rules unless overwritten via its own rule-specific section

• each name (e.g., here: queue, name nCPUs, memory, maxTime, output, and error) must be defined in the __default__ section of the cluster configuration file
• note that in this example, we provided some extra parameters for convenience such as `name` (so the cluster job will have a reasonable name and can be recognized) that are not strictly necessary
• the `{threads}` syntax of the `nCPUs` name can be generally used and is a placeholder for the specified number of threads for the particular rule, as specified in the corresponding Snakefile
• in our example, memory is given in Megabytes, so 5000 refers to roughly 5 GB. Queue names are either `htc` or `1day`. Adjust this accordingly to your cluster system.
• for more details, see the Snakemake documentation

Note: From a practical point of view, just try to mimic the parameters that you usually use for your cluster system, and modify the cluster configuration file accordingly. For example, if you need an additional argument such as `-A` (which stands for the `group` you are in for a SLURM-based system), simply add `-A {cluster.group}` to the command line call and add a `group` parameter to the `__default__` section (see also the note below).

### 9.4 Frequently asked questions

Here are a few typical use cases, which we will extend regularly in the future if the need arises:

1. I received an error, and the pipeline did not finish.
   
   As explained in Section *Handling errors*, you first have to identify and fix the error. Rerunning then becomes trivially easy: just restart *Snakemake*, it will start off where it left off: at the step that produced that error.

2. I received an error, and I do not see any error message.
   
   First, check the cluster output and error files if you run *diffTF* in cluster mode. They mostly contain an actual error message or at least the print the exact command that resulted in an error. If you executed locally or still cannot find the error message, see below for guidelines.

3. I want to rerun a specific part of the pipeline only.
   
   This common scenario is also easy to solve: Just invoke *Snakemake* with `--forcerun {rulename}`, where `{rulename}` is the name of the rule as defined in the Snakefile. *Snakemake* will then rerun the specified run and all parts downstream of the rule. If you want to avoid rerunning downstream parts (think carefully about it, as there might be changes from the rerunning that might have consequences for downstream parts also), you can combine `--forcerun` with `--until` and specify the same rule name for both.

4. I want to modify the workflow.
   
   Simply add or modify rules to the Snakefile, it is as easy as that.

5. *diffTF* finished successfully, but nothing is significant.
   
   This can and will happen, depending on the analysis. The following list provides some potential reasons for this:

   • The two conditions are in fact very similar and there is no signal that surpasses the significance threshold. You could, for example, check in a PCA plot based on the peaks that are used as input for *diffTF* whether they show a clear signal and separation.

   • There is a confounding factor (like age) that dilutes the signal. One solution is to add the confounding variable into the design model, see above for details. Again, check in a PCA plot whether samples cluster also according to another variable.
• You have a small number of samples or one of the groups contains a small number of samples. In both cases, if you run the permutation-based approach, the number of permutations is small, and there might not be enough permutations to achieve significance. For example, if you run an analysis with only 10 permutations, you cannot surpass the 0.05 significance threshold. As a solution, you may switch to the analytical version. Be aware that this requires to rerun large parts of the pipeline from the `diffPeaks` step onwards.

• You have a very small number of peaks and therefore also a small number of TF binding sites within the peaks, resulting in many TFs to be skipped in the analysis due to an insufficient number of binding sites. As a solution, try increasing the number of peaks. We recommend having at least a few thousand peaks, but this can hardly be generalized and depends too much on the biology, the size of the peaks etc.

6. I want to change the value of a parameter.

   Coming soon

   If you feel that a particular use case is missing, let us know and we will add it here!
10.1 Error types

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

- **Temporary errors** (often when running in a cluster setting)
  - might occur due to temporary problems such as bad nodes, file system issues or latencies
  - rerunning usually fixes the problem already. Consider using the option `--restart-times` in Snakemake.

- **Permanent errors**
  - indicates a real error related to the specific command that is executed
  - rerunning does not fix the problem as they are systematic (such as a missing tool, a library problem in R)

From our experience, most errors occur due to the following issues:

- Software-related problems such as R library issues, non-working conda installation etc. Consider using the Singularity-enhanced version of `diffTF` (version 1.2 and above) that immediately solves these issues.

- issues arising from the data itself. Here, it is more difficult to find the cause. We tried to cover all cases for which `diffTF` may fail, so please post an issue on our Bitbucket Issue Tracker if you believe you found a new problem.

10.2 Identify the cause

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 appears on the console. Check the output before the line “Error in rule”, and try to identify what went wrong. Errors from R script should in addition be written to the corresponding R log files in the in the `LOGS_AND_BENCHMARKS` directory. Sometimes, no error message might be displayed, and the output may look like this:
Error in rule intersectTFBSAndBAM:
jobid: 1287
   extension100/FL-WTvsFL-EKO.all.HXA10.allBAMs.overlaps.bed, output-FL-WT-vs-EKO-
   ATAC-distal-Linj-activ/TF-SPECIFIC/HXA10/extension100/FL-WTvsFL-EKO.all.HXA10.
allBAMs.overlaps.bed.gz, output-FL-WT-vs-EKO-ATAC-distal-Linj-activ/TEMP/
   extension100/FL-WTvsFL-EKO.all.HXA10.allTFBS.peaks.extension.saf

RuleException:
CalledProcessError in line 493 of /mnt/data/bioinfo_tools_and_refs/bioinfo_tools/
diffTF/src/Snakefile:
Command ' set -euo pipefail; ulimit -n 4096 &&
zgrep "HXA10_TFBS\." output-FL-WT-vs-EKO-ATAC-distal-Linj-activ/TEMP/
extension100/FL-WTvsFL-EKO.all.allTFBS.peaks.extension.bed.gz | awk 'BEGIN {
   OFS = "\" " } { print $4"_"$2"_"$3,$1,$2,$3,$6}' | sort -u -k1,1 >output-FL-WT-
vs-EKO-ATAC-distal-Linj-activ/TEMP/extension100/FL-WTvsFL-EKO.all.HXA10.allTFBS.
   peaks.extension.saf &&
   featureCounts -F SAF -T 4 -Q 10
   -o a output-FL-WT-vs-EKO-ATAC-distal-Linj-activ/TEMP/
   extension100/FL-WTvsFL-EKO.all.HXA10.allTFBS.peaks.extension.saf
   -s -o a output-FL-WT-vs-EKO-ATAC-distal-Linj-activ/TF-
   SPECIFIC/HXA10/extension100/FL-WTvsFL-EKO.all.HXA10.allBAMs.overlaps.bed
   -a /mnt/data/common/tobias/diffTF/ATAC-bam-files/FL-Wt-ProB-1.bam /mnt/data/
   common/tobias/diffTF/ATAC-bam-files/FL-Wt-ProB-2.bam /mnt/data/common/tobias/
   diffTF/ATAC-bam-files/FL-Wt-ProB-3.bam /mnt/data/common/tobias/diffTF/ATAC-bam-
   files/FL-Ebf1-KO-ProB-1.bam /mnt/data/common/tobias/diffTF/ATAC-bam-files/FL-
   Ebf1-KO-ProB-2.bam /mnt/data/common/tobias/diffTF/ATAC-bam-files/FL-Ebf1-KO-
   ProB-3.bam &&
gzip -f < output-FL-WT-vs-EKO-ATAC-distal-Linj-activ/TF-SPECIFIC/
   HXA10/extension100/FL-WTvsFL-EKO.all.HXA10.allBAMs.overlaps.bed > output-FL-WT-
   vs-EKO-ATAC-distal-Linj-activ/TF-SPECIFIC/HXA10/extension100/FL-WTvsFL-EKO.all.
HXA10.allBAMs.overlaps.bed.gz ' returned non-zero exit status 1.

Finding the exact error can be troublesome, and we recommend the following:

- execute the exact command as pasted above in a stepwise fashion. The command above consists of several
  commands that are chained together with &&, so copy and paste the individual parts, starting with the first
  part, execute it locally, and see if you receive any error message.
- once you have an error message, you can start troubleshooting it. The first step is always to actually see
  and understand the error.

  • in cluster mode: either error, output or log file of the corresponding rule that threw the error in the
    LOGS_AND_BENCHMARKS directory. If you are unsure in which file to look, identify the rule name that caused
    the error and search for files that contain the rule name in it.

In both cases, you can check the log file that is located in .snakemake/log. Identify the latest log file (check the
date), and then either open the file or use something along the lines of:
This is particularly helpful if the Snakemake output is long and you have troubles identifying the exact step in which an error occurred.

## 10.3 Common errors

We here provide a list of some of the errors that can happen and that users reported to us. This list will be extended whenever a new problem has been reported.

1. **R related problems**

   Many errors are R related. R and Bioconductor use a quite complex system of libraries and dependencies, and you may receive errors that are related to R, Bioconductor, or specific libraries.

   ```
   *** caught segfault ***
   ...
   Segmentation fault
   ...
   
   Note: This particular message may also be related to an incompatibility of the DiffBind and DESeq2 libraries. See the changelog for details, as this has been addressed in version 1.1.5.
   ```

   More generally, however, such messages point to a problem with your R and R libraries installation and have per se nothing to do with diffTF. In such cases, we advise to reinstall the latest version of Bioconductor and ask someone who is experienced with this to help you. Unfortunately, this issue is so general that we cannot provide any specific solutions. To troubleshoot and identify exactly which library or function causes this, you may run the R script that failed in debug mode and go through it line by line. See the next section for more details.

   ```
   Note: We strongly recommend running the Singularity version of diffTF (version 1.2 and above) that immediately solves these issues. See the Change log for more details and the section Try it out now!
   ```

## 10.4 Fixing the error

### 10.4.1 General guidelines

After locating the error, fix it accordingly. We here provide some guidelines of different error types that may help you fixing the errors you receive:

- **Errors related to erroneous input**: These errors are easy to fix, and the error message should be indicative. If not, please let us know, and we improve the error message in the pipeline.

- **Errors of technical nature**: Errors related to memory, missing programs, R libraries etc can be fixed easily by making sure the necessary tools are installed and by executing the pipeline in an environment that provides the required technical requirements. For example, if you receive a memory-related error, try to increase the available memory. In a cluster setting, adjust the cluster configuration file accordingly by either increasing the default memory or (preferably) or by overriding the default values for the specific rule.
• Errors related to Snakemake: In rare cases, the error can be due to Snakemake (corrupt metadata, missing files, etc). If you suspect this to be the case, you may delete the hidden .snakemake directory in the folder from which you started the analysis. Snakemake will regenerate it the next time you invoke it then.

• Errors related to the input data: Error messages that indicate the problem might be located in the data are more difficult to fix, and we cannot provide guidelines here. Feel free to contact us.

10.4.2 Debugging R scripts to identify the cause of an error

If an R script fails with a technical error such as caught segfault (a segmentation fault), you may want to identify the library or function call that causes the message in order to figure out which library to reinstall. To do so, open the R script that fails in RStudio, and execute the script line by line until you identify the line that causes the issue. Importantly, read the instructions in the section at the beginning of the script that is called SAVE SNAKEMAKE S4 OBJECT THAT IS PASSED ALONG FOR DEBUGGING PURPOSES. Briefly, you simply have to make the snakemake object available in your R workspace, which contains all necessary information to execute the R script properly. Normally, Snakemake automatically loads that when executing a script. To do so, simply execute the line that is pasted there in R, it is something like this:

```r
snakemake = readRDS("{outputFolder}/LOGS_AND_BENCHMARKS/checkParameters.R.rds")
```

Replace `{outputFolder}` by the folder you used for the analysis, and adjust the `checkParameters` part also accordingly. Essentially, you just have to provide the path to the corresponding file that is located in the `LOGS_AND_BENCHMARKS` subdirectory within the specified output directory.

10.4.3 Rerunning Snakemake

After fixing the error, rerun Snakemake. Snakemake will continue at the point at which the error message occurred, without rerunning already successfully computed previous steps (unless specified otherwise).
Biological motivation

Transcription factor (TF) activity constitutes an important readout of cellular signalling pathways and thus for assessing regulatory differences across conditions. However, current technologies lack the ability to simultaneously assessing activity changes for multiple TFs and surprisingly little is known about whether a TF acts as repressor or activator. To this end, we introduce the widely applicable genome-wide method diffTF to assess differential TF binding activity and classifying TFs as activator or repressor by integrating any type of genome-wide chromatin with RNA-Seq data and in-silico predicted TF binding sites.

For a graphical summary of the idea, see the section Workflow

We also put the paper on bioRxiv, please see the section Citation for details.
We provide a toy dataset that is included in the Git repository to test diffTF. It is a small ATAC-Seq/RNA-Seq dataset comparing two cell types along the hematopoietic differentiation trajectory in mouse (multipotent progenitors - MPP - versus granulocyte-macrophage progenitors - GMP) and comes from Rasmussen et al. 2018. Generally, hematopoiesis is organized in a hierarchical manner, and the following Figure shows the hematopoietic hierarchy in more detail and also places GMP and MPP cells:

![Hematopoietic hierarchy](http://www.bloodjournal.org/content/bloodjournal/125/23/3542/F1.large.jpg)

In the example analysis, you can investigate the differential TF activity of a set of 50 (or even all of the 400+) TFs to identify the known drivers of the well-studied mouse hematopoietic differentiation system. Overall, we expect to see TFs that more specific for stem cells renewal being more active in the MPPs, while in GMPs, TFs responsible for the further myeloid cell differentiation (CEBP family, NFIL3) should be enriched.
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 have questions, doubts, ideas or problems, please use the Bitbucket Issue Tracker. We will respond in a timely manner.
If you use this software, please cite the following reference:


We also put the paper on *bioRxiv*, please read all methodological details here: *Quantification of differential transcription factor activity and multiomic-based classification into activators and repressors: diffTF*. 
CHAPTER 15

Change log

Version 1.2.5 (2019-03-13)

• Updated the TFBS_hg38_FIMO_HOCOMOCOv11 archive one more time to exclude non-assembled contigs such as HLA*. To make the pipeline more stable for such edge cases, the parameter `dir_TFBS_sorted` has been removed, and sorting and filtering of chromosomes is now always performed. Only chromosomes are kept in both the consensus peak files and the TFBS bed files that start with `chr` and are neither sex chromosomes (`chrX` or `chrY`) nor `chrM`. If you want to keep sex chromosomes in your analysis (although we think this is not recommended), simply edit the Snakefile and remove the “`chrX`” and “`chrY`” occurrences in the two filtering rules.

Version 1.2.4 (2019-03-04)

• Fixed an issue with `checkParameterValidity.R` that caused an error message when loading TFBS files with a numeric score. Thanks to Scott Berry for pointing it out.

• Updated the TFBS_hg38_FIMO_HOCOMOCOv11 archive. The bed files are now properly pre-sorted.

Version 1.2.3 (2019-02-27)

• Added a pre-compiled list of 768 human TF with in-silico predicted TFBS based on the HOCOMOCO 11 database and PWMScan for hg38 as well as updating the other pre-compiled lists to account for recent changes and retractions in the HOCOMOCO database. See section `_parameter_dir_TFBS` for details.

• added an additional filtering in the binning step for a rare corner case due to changes in the number of samples during an analysis.

Version 1.2.2 (2019-02-01)

• Minor code fixed. Removed the creation of the circular plot, which has been replaced with the Volcano plot over time. Fixed a bug that could have led to wrong log2 fold-change values for the RNA-Seq data under special circumstances. We recommend rerunning the `summaryFinal` rule. Ask us for more details if you are concerned about this.

Version 1.2.1 (2019-01-22)

• Increased the value for `expressions` in R from 5000 (the R default) to 10000. Some users reported that they receive a “`Error: evaluation nested too deeply: infinite recursion / options(expressions=)?`” error message. Thanks to Benedict Man Hung Choi!
Version 1.2 (2018-12-10)
• The Snakemake / \textit{diffTF} pipeline can now be combined with \textit{Singularity}. Singularity is similar to Docker and provides a containerization approach. This has significant implications for users: Except for Snakemake and Singularity, no other tool, R or R package has to be installed prior to using \textit{diffTF} anymore, which makes installing \textit{diffTF} much easier and completely independent of the underlying operating system. We now provide two Singularity containers with all necessary tools and packages that are automatically integrated into the workflow. See the section \textit{Adaptations and notes when running with Singularity} and \textit{Try it out now!} for more details. \textbf{Please note that for this to work reliably, Snakemake must be updated to at least version 5.3.1.}
• added two validity checks in the \texttt{checkParameterValidity.R} script for the TFBS files. Start and end coordinates are now asserted whether they are non-negative.

Version 1.1.8 (2018-11-07)
• changed the call to the \texttt{mlv} function from the \textit{modeest} package due to a breaking implementation change in version 2.3.2 that was published end of October 2018. \textit{diffTF} now checks the package version for \textit{modeest} and calls the functions in dependence of the specific version.

Version 1.1.7 (2018-10-25)
• the default value of the minimum number of data points for a CG bin to be included has been raised from 5 to 20 to make the variance calculation more reliable
• various small updates to the \texttt{summaryFinal.R} script

Version 1.1.6 (2018-10-11)
• fixed small issue in \texttt{checkParameterValidity.R} when not having sufficient permissions for the folder in which the fasta file is located
• updated the \texttt{summaryFinal.R} script. Now, for the Volcano plot PDF, in addition to adj. p-values, also the raw p-values are plotted in the end. This might be helpful for datasets with small signal when no adj. p-value is significant. In addition, labeling of TFs is now skipped when the number of TFs to label exceeds 150. This makes the step faster and the PDF smaller and less crowded.
• small updates to the translation table for mm10
• adding two local rules to the Snakefile for potential minor speed improvements when running in cluster mode

Version 1.1.5 (2018-08-14)
• optimized \texttt{checkParameterValidity.R} script, only TFBS files for TFs included in the analysis are now checked
• addressed an R library compatibility issue independent of \textit{diffTF} that users reported. In some cases, for particular versions of R and Bioconductor, R exited with a \texttt{segfault} (memory not mapped) error in the \texttt{checkParameterValidity.R} that seems to be caused by the combination of \textit{DiffBind} and \textit{DESeq2}. Specifically, when \textit{DiffBind} is loaded \textit{before} \textit{DESeq2}, R crashes with a segmentation fault upon exiting, whereas loading \textit{DiffBind} \textit{after} \textit{DESeq2} causes no issue. If there are further issues, please let us know. Thanks to Gyan Prakash Mishra, who first reported this.
• fixed an issue when the number of peaks is very small so that some TFs have no overlapping TFBS at all in the peak regions. This caused the rule \texttt{intersectTFBSAndBAM} to exit with an error due to grep’s policy of returning exit code 1 if no matches are returned (thanks to Jonas Ungerbeck, again).
• removed the \texttt{--timestamp} option in the helper script \texttt{startAnalysis.sh} because this option has been removed for Snakemake >5.2.1
• Documentation updates
Version 1.1.4 (2018-08-09)
  • minor, updated the checkParameterValidity.R script and the documentation (one package was not mentioned)

Version 1.1.3 (2018-08-06)
  • minor, fixed a small issue in the Volcano plot (legends wrong and background color in the plot was not colored properly)

Version 1.1.2 (2018-08-03)
  • fixed a bug that made the 3.analyzeTF.R script fail in case when the number of permutations has been changed throughout the analysis or when the value is higher than the actual maximum number (thanks to Jonas Ungerbeck)

Version 1.1.1 (2018-08-01)
  • Documentation updates (referenced the bioRxiv paper, extended the section about errors)
  • updated the information on how to load the snakemake object into the R workspace in the corresponding R scripts
  • fixed a bug that made the labels in the Volcano plot switch sides (thanks to Jonas Ungerbeck)
  • merged some diagnostic plots for the AR classification in the last step
  • renamed R scripts and R log files to make them consistent with the cluster output and error files

  • added a new parameter dir_TFBS_sorted in the config file to specify that the TFBS input files are already sorted, which saves some computation time by not resorting them
  • updated the TFBS files that are available via download (some files were not presorted correctly)
  • added support for single-end BAM files. There is a new parameter pairedEnd in the config file that specifies whether reads are paired-end or not.
  • restructured some of the permutation-related output files to save space and computation time. The rule concatenateMotifsPerm should now be much faster, and the TF-specific ...outputPerm.tsv.gz files are now much smaller due to an improved column structure

  • fixed a bug in 2.DiffPeaks.R that sometimes caused the step to fail, thanks to Jonas Ungerbeck for letting us know
  • fixed a bug in 3.analyzeTF for rare corner cases when DESeq fails

Version 1.0 (2018-07-01)
  • released stable version
CHAPTER 16

License

diffTF is licensed under the MIT License:

Copyright (c) 2018 Christian Arnold (carnold@embl.de) & Ivan Berest (berest@embl.de)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.