

Documentation for

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

Ivan Berest^{1*}, Christian Arnold^{1*}, Armando Reyes-Palomares¹, Kasper Dindler
Rasmussen^{2,3}, Kristian Helin^{2,3} & Judith B. Zaugg¹

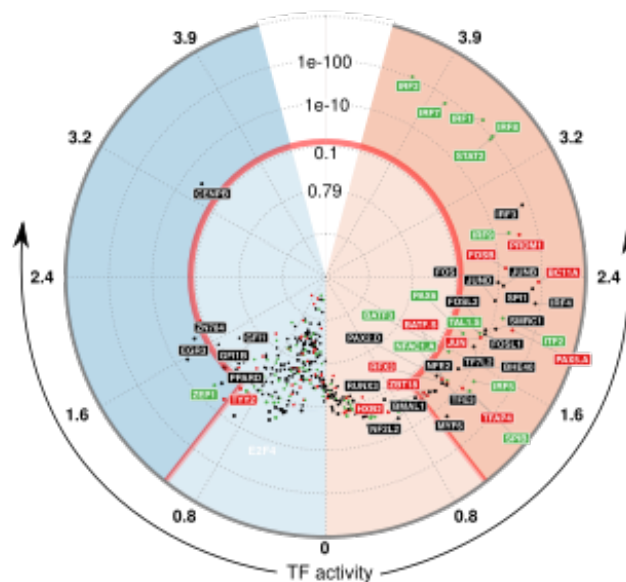
¹ *EMBL-European Molecular Biology Laboratory, Heidelberg, Germany*

² *Biotech Research and Innovation Centre (BRIC), University of Copenhagen, Copenhagen*

³ *Novo Nordisk Foundation Center for Stem Cell Biology, Copenhagen*

** shared first authorship*

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



Last update: 11/29/17

This document provides documentation and additional information for the *diffTF* pipeline.

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 Issue Tracker at <https://git.embl.de/grp-zaugg/diffTF/issues>. We will respond in a timely manner.

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

Ivan Berest*, Christian Arnold*, Armando Reyes-Palomares, Kasper Dindler Rassmussen, Kristian Helin & Judith B. Zaugg. Genome-wide quantification of differential transcription factor activity: diffTF. 2017. submitted.

Table of Contents

1. Quick Start.....	4
2. Prerequisites.....	5
2.1. Snakemake.....	5
2.2. R and R packages.....	5
2.3. samtools, bedtools, Subread.....	5
3. Running your own analysis.....	6
4. Pipeline Details.....	7
4.1. Input.....	7
4.2. Workflow.....	7
4.3. General configuration file for the analysis (config.json).....	8
4.3.1. Section "par_general".....	9
4.3.2. Section "samples".....	11
4.3.3. Section "peaks".....	11
4.3.4. Section "additionalInputFiles".....	12
4.4. Input metadata.....	14
4.5. Structure and Content of the Output Folders and Files.....	15
4.5.1 Folder FINAL_OUTPUT.....	15
4.5.2 Folder PEAKS.....	18
4.5.3 Folder TF-SPECIFIC.....	19
4.5.4 Folder LOGS_AND_BENCHMARKS.....	20
4.5.5 Folder TEMP.....	20
4.6. Working with the pipeline.....	21
4.7. Handling errors.....	22

1. Quick Start

First, thank you for the interest in *diffTF*! As mentioned above on page 2, 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. **If you run into troubles or questions, make sure to carefully read this document and Section 4 in particular.**

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

1. Install the necessary tools (Snakemake, samtools, bedtools, and subread; see Section 2). We recommend installing them via conda, in which case the installation is as easy as

```
$ conda install -c bioconda snakemake bedtools samtools subread
```

If conda is not yet installed, follow the instructions in <https://conda.io/docs/user-guide/install/index.html>. If you want to install the tools manually and outside of the conda framework, see Section 2.

2. Clone the Git repository:

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

3. To run the example analysis for 50 TF, simply perform the following steps::

- a) Change into the example/input directory within the Git repository

```
$ cd diffTF/example/input
```

- b) Download the data via the download script

```
$ sh downloadAllData.sh
```

- c) To test if the setup is correct, start a dryrun via the helper script *startAnalysisDryRun.sh*

```
$ sh startAnalysisDryRun.sh
```

- d) Once the dryrun is successful, start the analysis via the helper script *startAnalysis.sh*

```
$ sh startAnalysis.sh
```

4. To run your own analysis, modify the files *config.json* and *sampleData.tsv* accordingly.. See the instructions in Section 3 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 into Section 4 to troubleshoot.

2. Prerequisites

2.1. Snakemake

Please ensure that you have at least version 4.3 installed. Principally, there are multiple ways to install Snakemake, see http://snakemake.readthedocs.io/en/stable/getting_started/installation.html.

As outlined in Section 1, we recommend installing it via conda

2.2. 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:

```
$ install.packages(c("checkmate", "futile.logger", "tidyverse",  
"reshape2", "gridExtra", "scales", "jsonlite", "RcolorBrewer", "rlist",  
"ggrepel", "lsr", "modeest", "locfdr", "boot"))  
  
$ source("https://bioconductor.org/biocLite.R")  
  
$ biocLite(c("limma", "vsn", "csaw", "DESeq2", "DiffBind", "geneplotter",  
"Rsamtools"))
```

2.3. samtools, bedtools, Subread

In addition, samtools (<http://www.htslib.org/download/>), bedtools (<http://bedtools.readthedocs.io>) and Subread (<http://subread.sourceforge.net/>) are needed to run *diffTF*. As outlined in Section 1, we recommend installing them via conda.

3. Running your own analysis

Running your own analysis is almost as easy as running the example analysis. 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 4 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 *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)

Important notes:

- For Snakemake to run properly, the R folder with the scripts has to be in the same folder as the Snakefile.
- Since running the pipeline might be computationally demanding, 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. Also, adjust the number of available cores accordingly. The pipeline can be invoked in a highly parallelized manner, so the more cores are available, the better!
- The pipeline is written in Snakemake, so for a deeper understanding and troubleshooting errors, some knowledge of Snakemake is invaluable. The same holds true for running the pipeline in a cluster setting. We recommend using a proper cluster configuration file in addition. For guidance and user convenience, we provide different cluster configuration files for a small (up to 10-15 samples) and large (>15 samples) analysis. See the folder *src/clusterConfigurationTemplates* for examples. Note that the sample number guidelines above are very rough estimates only. See the Snakemake documentation for details for how to use cluster configuration files.

4. Pipeline Details

In this section, we explain all relevant details for the *diffTF* pipeline. If you feel something is missing, do not hesitate to contact us.

4.1. Input

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

- BAM file with aligned reads for each sample (see Section 4.2.4, parameter *summaryFile*)
- genome reference fasta that has been used to produce the BAM files (see Section 4.2.4, parameter *refGenome_fasta*)
- Optionally: corresponding RNA-Seq data (see Section 4.2.4, parameter *RNASeqCounts*)

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

- TF-specific list of TFBS (see Section 4.2.4, parameter *dir_TFBS*)
- HOCOMOCO mapping table (see Section 4.2.4, parameter *HOCOMOCO_mapping*)

Note that we currently do not yet provide TFBS for hg38, but we are working on them!

4.2. Workflow

Figure 1 shows a schematic of the diffTF workflow, with input and output of the pipeline highlighted.

Figure 2 shows the exact workflow (a so-called directed acyclic graph, or DAG) that is executed when calling Snakemake. Each node represents a rule name as defined in the Snakefile, and each arrow a dependency. The example shown is for two TFs, CEBPB and CTCF for the two samples GMP.WT1 and MPP.WT1.

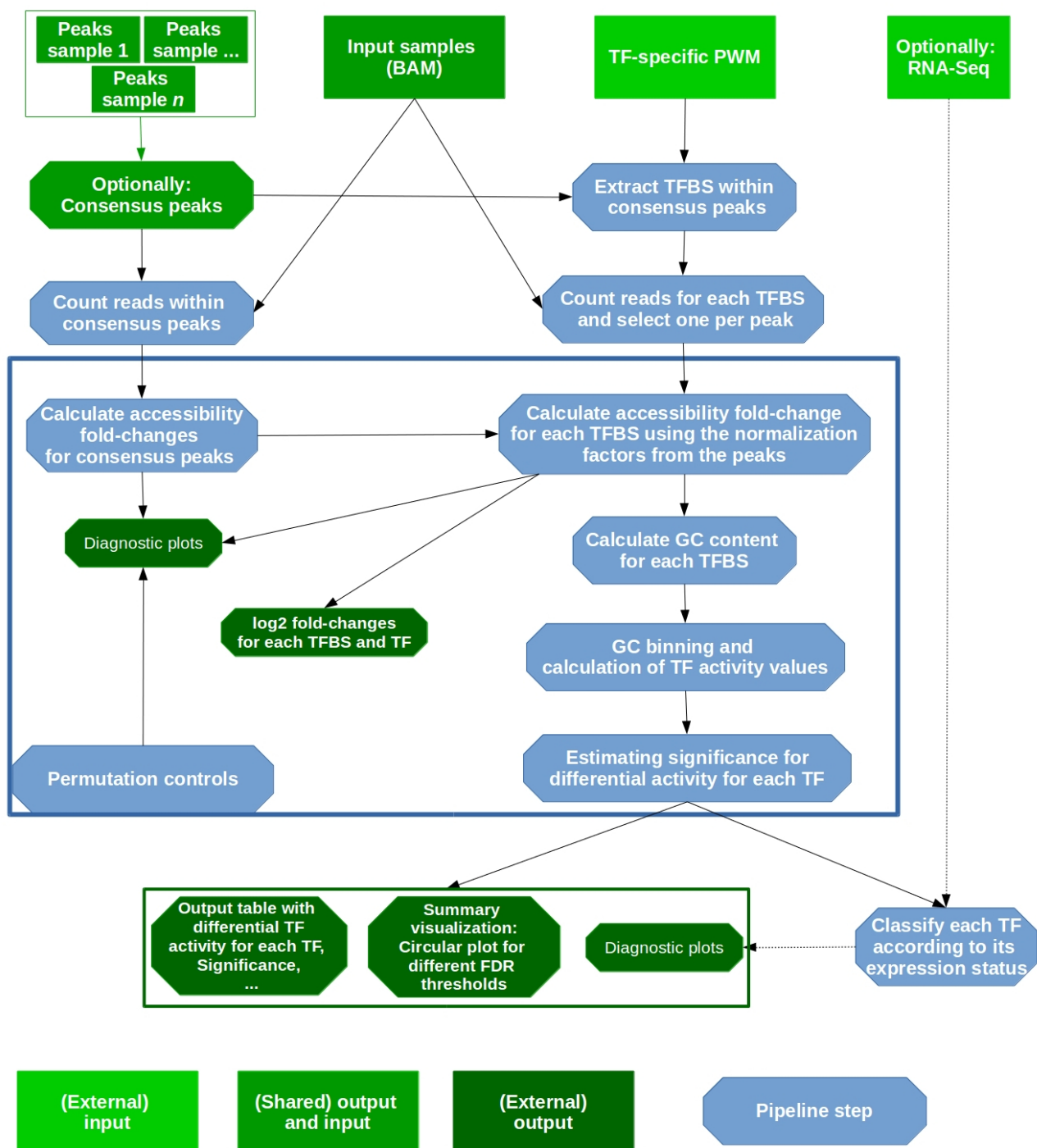


Figure 1. Workflow of diffTF. For more details, see the Online Methods.

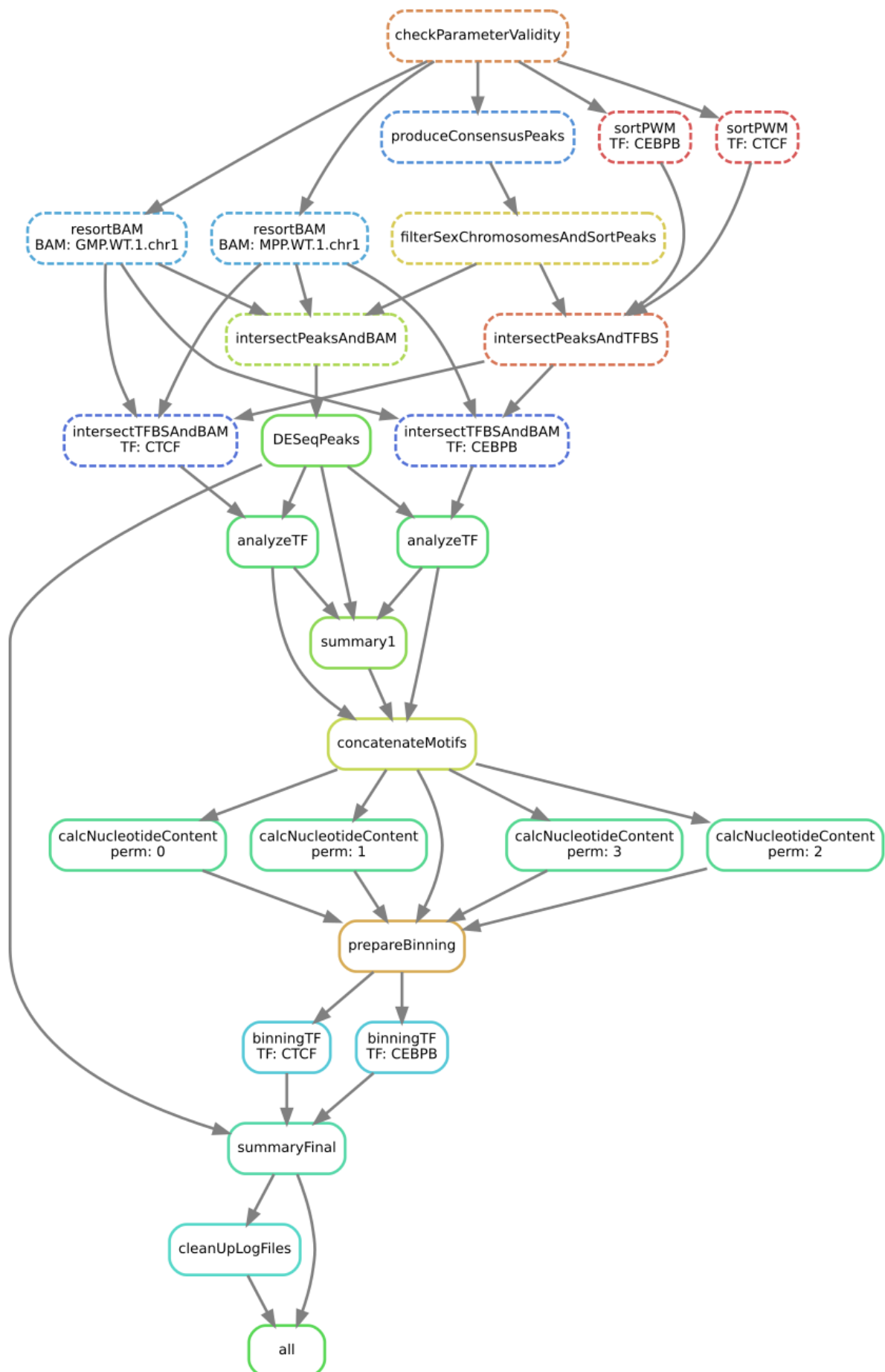


Figure 2. Directed acyclic graph of a small example as used in Snakemake

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

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

Please note the following important points:

- **neither section nor parameter names must be changed.**
- **For parameters that specify a path, both absolute and relative paths are possible. We recommend specifying an absolute path. Relative paths must be specified relative to the Snakemake working directory.**
- **For parameters that specify a directory, there should be no trailing slash.**

In the following, we explain all parameters in detail, organized by section names.

4.3.1. 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.

designContrast

String. Default “~ Treatment + conditionSummary”. Design formula for the differential accessibility analysis in DESeq2.

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 wildtype, 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.

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”.

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 circular 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.

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. Specifically, for the condition with fewer samples, 50% will be randomly chosen and switched with the same number of samples from the other condition. This procedure maximizes randomized conditions. Note that the running time of the pipeline will be increased when using permutations because parts of the pipeline are run for each permutation.

TFs

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* will be used. 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 that for each TF {name}, a file “{name}_TFBS.bed” needs to be present in the directory *dir_TFBS*.

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

dir_scripts

String. 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. Note that RNA-Seq integration is only included in the very last step of the pipeline, so it can also be easily integrated later.

4.3.2. Section "samples"

summaryFile

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

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.

4.3.3. 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 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:

4. Identifier (will be made unique for each if this is not the case already)
5. Score
6. Strand

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. All individual peak files must be in the same format. See the help for DiffBind *dba* for a full list of supported formats, the most common include:

- "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 (from MACS2)

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*.

4.3.4. 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. Note that 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.

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.

dir TFBS

String. Path to the directory where the TF-specific files for TFBS results are stored. Each 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 these files as described in the publication as a separate download:

- hg19: For a pre-compiled list of 620 human TF with in-silico predicted TFBS based on the HOCOMOCO 10 database and PWMScan for hg19, download this file:
https://www.embl.de/download/zaugg/diffTF/TFBS/TFBS_hg19_PWMScan_HOCOMOCO_v10.tar.gz
- hg38: For a pre-compiled list of 771 human TF with in-silico predicted TFBS based on the HOCOMOCO 11 database and FIMO from the MEME suite¹ for hg38, download this file:
https://www.embl.de/download/zaugg/diffTF/TFBS/TFBS_hg38_FIMO_HOCOMOCOv11.tar.gz
- mm10: For a pre-compiled list of 423 mouse TF with in-silico predicted TFBS based on the HOCOMOCO 10 database and PWMScan for mm10, download this file:
https://www.embl.de/download/zaugg/diffTF/TFBS/TFBS_mm10_PWMScan_HOCOMOCOv10.tar.gz

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.

¹ Bailey, Timothy L., et al. "MEME SUITE: tools for motif discovery and searching." *Nucleic acids research* 37.suppl_2 (2009): W202-W208.

RNASeqCounts

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

If no RNA-Seq data is included, set to the empty string “”. Otherwise, if the 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 in

<https://labs.genetics.ucla.edu/horvath/CoexpressionNetwork/Rpackages/WGCNA/faq.html>,

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*). If you have RNA-Seq data for only a subset of the input samples, this is no problem – the classification will then anturally 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 the parameter *HOCOMOCO_mapping*.

HOCOMOCO_mapping

String. Path to the TF-Gene translation table.

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 and (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.

4.4. 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. 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 with a “chr” as prefix. The pipeline may crash if the “chr” part is missing.
- **“peaks”**: absolute path to the sample-specific peak file, in the format as given by the 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, ...)
- If applicable, all additional variables from the design formula except *conditionSummary* must also be present as a separate column.

4.5. Structure and Content of the Output Folders and Files

The pipeline produces quite a large number of output files, only some of which are however relevant for the regular user. 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 the parameters *comparisonType* and *regionExtension* as specified in the configuration file.

Most files have one of the following file formats:

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

4.5.1 Folder FINAL_OUTPUT

- extension{regionExtension}: Stores results related to the user-specified extension size (parameter *regionExtension*)
 - {comparisonType}.allMotifs.tsv.gz: Summary table for each TFBS with the columns as follows:

- permutation: The number of the permutation. Note that 0 always refers to the non-permuted, real data, while permutations > 0 reflect real permutations.
- TF: name of the TF
- chr, MSS, MES, strand, TFBSID : Genomic location and identifier of the (extended) TFBS
- PSS, PES, peakID: Genomic location and annotation of the overlapping peak region
- baseMean, log2FoldChange , lfcSE, stat, pvalue, padj: Results from the DESeq2 analysis. See the DESeq2 documentation for details.
- *{comparisonType}.TF_vs_peak_distribution.tsv*: . The columns are as follows:
 - TF: name of the TF
 - permutation: The number of the permutation. Note that 0 always refers to the non-permuted, real data, while permutations > 0 reflect real permutations.
 - 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) p-value of the t-test
 - T_statistic: the value of the T statistic from the t-test
 - TFBS_num: number of TFBS
 - 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
- *{comparisonType}.summary.tsv*: The final summary table that is also used for the final circular visualization (see below). The columns are as follows:
 - TF: name of the TF
 - permutation: The number of the permutation. Note that 0 always refers to the non-permuted, real data, while permutations > 0 reflect real permutations.
 - 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 circular plot. Without permutations, this value is shown, while for permutations, the weighted_meanDifference_enrichment is used.
 - weighted_Tstat: the weighted value of the T statistic across all CG bins.
 - TFBS: The number of TF binding sites that overlap with the peaks
 - variance: Estimated variance for the *weighted_Tstat* estimate that is incorporated to calculate the p-value
 - weighted_meanDifference_enrichment: If permutations are used, the enrichment over the background is calculated based on the weighted_meanDifference values for

the real and permuted data, respectively. Specifically, by default, the minimum and maximum of the permuted `weighted_meanDifference` values is taken as permutation thresholds, and the real `weighted_meanDifference` values are then divided by these thresholds to calculate an enrichment.

- `weighted_Tstat_centralized`: a centralized version of the `weighted_Tstat` values that is used to calculate raw p-values while including the variance also. If possible, the MLE estimate of the distribution median is used for centralization.
- `pvalue`: raw p-values
- `pvalueAdj`: adjusted p-values (FDR, Benjamini & Hochberg (1995) correction)
- `yValue`: the y-value for the plotting
- `median.cor.tfs`: the median correlation, related to the RNA-Seq classification
- `classification`: RNA-Seq classification (either activator, undetermined, repressor or not-expressed)
- `Cohend_factor`, `weighted_CD`, `weighted_median`, `weighted_sd`: Not used.
- *{comparisonType}.summary.circular.pdf*: The final visualization of the *diffTF* results. The PDF contains multiple pages, The PDF contains multiple pages, the structure of which varies depending on the parameters:
 - Number of permutations > 0
 - RNA-Seq integration
 - Pages 1-15: Visualization of the results including the permutations
 - Pages 16-30: Visualization of the results excluding the permutations

Within each of these two categories, the structure is identical (varying combinations of FDR threshold and RNA-Seq categories, see below)
 - No RNA-Seq integration:
 - Pages 1-5: Visualization of the results including the permutations for different FDR thresholds
 - Pages 6-10: Visualization of the results excluding the permutations for different FDR thresholds

Within each of these two categories, the structure is identical (varying combinations of FDR threshold and RNA-Seq categories, see below)
 - Number of permutations = 0
 - RNA-Seq integration: Pages 1-15 show the results in a format as described below for varying combinations of FDR threshold and RNA-Seq categories
 - No RNA-Seq integration: Pages 1-5 show the results for different FDR thresholds

If RNA-Seq data is integrated, different combinations of categories are shown on each page (1:activator-undetermined-repressor-not-expressed, 2:activator-undetermined-repressor, 3:activator-repressor), which along with different FDR thresholds gives rise to multiple individual plots. The values on the x-axis denote the effect size (if permutations are incorporated enrichment over background, otherwise the mean difference between the two conditions), with higher values indicating a higher differential TF activity between the two conditions. TFs that are similarly active between the two conditions are close to 0, while TFs more active in either condition are located in the left and right part of the plot. The y-axis (radial position) denotes the statistical significance (adjusted p-values). The significance threshold is indicated as red circular line. TFs that pass the significance threshold are labeled and, if RNA-Seq data is integrated, colored according to their predicted role (see above).

- *{comparisonType}.diagnosticPlots.pdf*: Various diagnostic plots for the final TF activity values. 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.

4.5.2 Folder PEAKS

Stores peak-associated files.

- if no consensus peak file was provided (parameter “*consensusPeaks*”):
{comparisonType}.consensusPeaks.bed and *consensusPeaks_lengthDistribution.pdf*: generated consensus peaks, before filtering (see below) as well as a diagnostic plot showing the length distribution of the peaks
- *{comparisonType}.consensusPeaks.filtered.sorted.bed*: Produced in rule *filterSexChromosomesAndSortPeaks*. Filtered consensus peaks (removal of peaks from one of the following chromosomes: chrX, chrY, chrM, chrUn*, *random*, *hap|_gl*
- *{comparisonType}.allBams.peaks.overlaps.bed*: Produced in rule *intersectPeaksAndBAM*. Counts for each consensus peak with each of the input BAM files
- *{comparisonType}.sampleMetadata.rds*: Produced in rule *DESeqPeaks*. Stores data for the input data (similar to the input sample table), for both the real data and the permutations.
- *{comparisonType}.peaks.rds*: Produced in rule *DESeqPeaks*. Stores all peaks that will be used in the analysis.
- *{comparisonType}.peaks.tsv*: Produced in rule *DESeqPeaks*. Stores the DESeq2 results of the differential accessibility analysis for the peaks.

- *{comparisonType}.normFacs.rds*: Produced in rule *DESeqPeaks*. Gene-specific normalization factors for each sample and peak. This file is produced after the differential accessibility analysis for the peaks. The normalization factors will be used for the TF-specific differential accessibility analysis.
- *{comparisonType}.diagnosticPlots.peaks.pdf* and *{comparisonType}.diagnosticPlots.peaks_permutation{perm}.pdf* for each permutation {perm}: Produced in rule *DESeqPeaks*. Various diagnostic plots for the differential accessibility peak analysis for the real and permuted data, respectively: (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 and (4) mean SD plots (row standard deviations versus row means)
- *{comparisonType}.DESeq.object.rds*: Produced in rule *DESeqPeaks*. The DESeq2 object from the differential accessibility peak analysis.

4.5.3 Folder TF-SPECIFIC

Stores TF-specific files.

for each TF {TF}:

- extension{regionExtension}
 - *{TF}.{comparisonType}.allBAMs.overlaps.bed.gz* and *{TF}.{comparisonType}.allBAMs.overlaps.bed.summary*: Overlap and *featureCounts* summary file of read counts across all TFBS for all input BAM files.
 - *{TF}.{comparisonType}.output.tsv*: Produced in rule *analyzeTF*. A summary table for the DESeq2 analysis. See the file *{comparisonType}.allMotifs.tsv.gz* in the *FINAL_OUTPUT* folder for a column description.
 - *{TF}.{comparisonType}.summary.rds*: Produced in rule *analyzeTF*. A summary table for the log2 fold-changes across all TFBS DESeq2 results.
 - *{TF}.{comparisonType}.diagnosticPlots.pdf* and *{TF}.{comparisonType}.diagnosticPlots_permutation{perm}.pdf*: Produced in rule *analyzeTF*. Various diagnostic plots for the differential accessibility TFBS analysis for the real and permuted data, respectively. See the description of the file *{comparisonType}.diagnosticPlots.peaks.pdf* in the *PEAKS* folder, which has an identical structure.
 - *{TF}.{comparisonType}.summaryPlots.pdf* and *{TF}.{comparisonType}.summaryPlots_permutation{perm}.pdf*: Produced in rule *analyzeTF*. A PDF with a summary of the DESeq2 analysis for the real and permuted data, respectively: Page 1 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. Page 2 shows the same but in a cumulative representation.
 - *{TF}.{comparisonType}.DESeq.object.rds*: Produced in rule *analyzeTF*. Original DESeq2 object.

- `{TF}.{comparisonType}.permutationResults.rds`: Produced in rule *binningTF*. An rds file containing a data frame that stores the results of bin-specific results.
- `{TF}.{comparisonType}.permutationSummary.tsv`: Produced in rule *binningTF*. A final summary table that summarizes the results across bins by calculating weighted means. The data of this table are used for the final visualization.
- `{TF}.{comparisonType}.covarianceResults.rds`: Produced in rule *binningTF*. An rds file containing a data frame that stores the results of the pairwise bin covariances and the bin-specific weights. Note that covariances are only computed for the real data but not the permuted ones.

4.5.4 Folder LOGS_AND_BENCHMARKS

Stores various log and error files.

- *.log files from R scripts: Each logfile is produced by the corresponding R script and contains debugging information as well as warnings and errors:
 - *1.produceConsensusPeaks.R.log*
 - *2.DESeqPeaks.R.log*
 - *3.analyzeTF.{TF}.R.log* for each TF {TF}
 - *4.summary1.R.log*
 - *5.prepareBinning.log*
 - *6.binningTF.{TF}.log* for each TF {TF}
 - *7.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*

4.5.5 Folder TEMP

Stores temporary and intermediate files.

- SortedBAM: Stores sorted versions of the original BAMs that are optimized for featureCounts.
 - `{basenameBAM}.bam` for each input BAM file: Produced in rule *resortBAM*. Resorted BAM file
- `extension{regionExtension}`: Stores results related to the user-specified extension size (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
- `{comparisonType}.{TF}.allTFBS.peaks.extension.saf` for each TF `{TF}`: Produced in rule *intersectTFBSAndBAM*. TF-specific file in SAF format needed for *featureCounts*
- *conditionComparison.rds*: Produced in rule *DESeqPeaks*. Stores the condition comparison as a string. Some steps in *diffTF* need this file as input.
- for each permutation `{perm}`:
 - `{comparisonType}.motifs.coord.permutation{perm}.bed.gz` and `{comparisonType}.motifs.coord.nucContent.permutation{perm}.bed.gz`: 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}.allTFData_processedForPermutations.rds` and `{comparisonType}.allTFUniqueData_processedForPermutations.rds`. Produced in rule *prepareBinning*, and needed subsequently for each *6.binningTF.R* step.
- `{comparisonType}.consensusPeaks.*`: Produced in rule *filterSexChromosomesAndSortPeaks*. Various temporary files related to the consensus peaks
- `{comparisonType}.checkParameterValidity.done`: temporary flag file
- `{TF}_TFBS.sorted.bed` for each TF `{TF}`: Produced in rule *sortPWM*. Coordinate-sorted version of the input TFBS.
- `{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

4.6. Working with the pipeline

diffTF is programmed as a Snakemake pipeline, which offers many advantages to the user because each step can easily be modified, parts of the pipeline can be rerun, and running the pipeline on different systems is easy with 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 Snakemake experience.

Here 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 4.5, 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 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.

3. I want to modify the workflow.

Simply add or modify rules to the Snakefile, it is as easy as that.

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

4.7. Handling errors

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`.
- 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)

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: either error, output or log file of the corresponding rule that threw the error

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 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.

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).