

---

**DROP**

**Apr 14, 2023**



---

## Contents:

---

<b>1</b>	<b>Installation</b>	<b>3</b>
1.1	Initialize a project . . . . .	4
1.2	Other DROP versions . . . . .	4
1.3	Installation without conda . . . . .	4
<b>2</b>	<b>Preparing the Input Data</b>	<b>7</b>
2.1	Config file . . . . .	7
2.2	Modularization of DROP . . . . .	13
2.3	Creating the sample annotation table . . . . .	14
2.4	Limiting FDR correction to subsets of genes of interest . . . . .	16
2.5	Files to download . . . . .	16
2.6	Example of RNA replicates . . . . .	17
2.7	Advanced options . . . . .	18
<b>3</b>	<b>Pipeline Commands</b>	<b>19</b>
3.1	Dry run . . . . .	19
3.2	Parallelizing jobs . . . . .	19
3.3	Executing subworkflows . . . . .	20
3.4	Rerunning the pipeline . . . . .	20
<b>4</b>	<b>Results and Output of DROP</b>	<b>23</b>
4.1	Aberrant Expression . . . . .	23
4.2	Aberrant Splicing . . . . .	24
4.3	Mono-allelic Expression . . . . .	25
4.4	RNA Variant Calling . . . . .	26
<b>5</b>	<b>License</b>	<b>27</b>
<b>6</b>	<b>Quickstart</b>	<b>29</b>



DROP is intended to help researchers use RNA-Seq data in order to detect genes with aberrant expression, aberrant splicing, mono-allelic expression, and RNA-Seq variant calling. It consists of 4 independent modules for each of those strategies. After installing DROP, the user needs to fill in the config file and sample annotation table (*Preparing the Input Data*). Then, DROP can be executed in multiple ways (*Pipeline Commands*).



# CHAPTER 1

---

## Installation

---

DROP is available on [bioconda](#) . In case the conda channel priority is set to `strict`, it should be reset to `flexible`:

```
conda config --set channel_priority true
```

We recommend using a dedicated conda environment (here: `drop_env`) for installing DROP. Use `mamba` instead of `conda` as it provides more reliable and faster dependency solving.

```
mamba create -n drop_env -c conda-forge -c bioconda drop --override-channels
```

In the case of installation troubles, we recommend using the fixed `DROP_<version>.yaml` installation file available on our [public server](#). Install the latest version and use the full path in the following command to install the conda environment `drop_env`

```
mamba env create -f DROP_1.3.3.yaml
```

Installation time: ~ 10min

We can test whether the pipeline runs through by setting up the demo dataset in an empty directory (e.g. `~/drop_demo`).

```
conda activate drop_env
mkdir ~/drop_demo
cd ~/drop_demo

# this command will download the necessary data and pipeline files
drop demo
```

DROP is run using `snakemake` commands.

Run time: ~25min

```
snakemake --cores 1 -n # dryrun
snakemake --cores 1
```

## 1.1 Initialize a project

The config and sample annotation files from the demo project can be modified to be used for a new project. Alternatively, a new DROP project can be set up using `drop init`.

```
cd <path/to/project>
drop init
```

This will create an empty `config.yaml` file that needs to be filled according to the project data. You also need to prepare a sample annotation file. Go to [Preparing the Input Data](#) for more details.

## 1.2 Other DROP versions

The following instructions are for users who have not used `conda` to install DROP previously. In order for the `pip` installation to take effect, you must first uninstall any previous installation using the following command. If you have not installed DROP previously, then there is no need to uninstall it.

```
pip uninstall drop
```

Other versions of DROP, such as `dev` can be found in the [repository](#) under different branches. Make sure that the [Installation without conda](#) are installed, preferably in a `conda` environment. Then install the desired version (e.g. `dev` in this example) from GitHub using `pip`.

```
pip install git+https://github.com/gagneurlab/drop.git@dev
```

Alternatively, you can clone the desired branch of the repository and install from directory.

```
git clone -b dev https://github.com/gagneurlab/drop.git
pip install ./drop
```

If the package needs to be updated frequently, it is more convenient to use the `-e` option of `pip`. Any new update pulled from the repository will be available without reinstall. Note, that this requires an explicit call to update any existing project ([Updating DROP](#)).

```
pip install -e ./drop

# update project directory
cd <path/to/project>
drop update
```

## 1.3 Installation without conda

The easiest way to ensure that all dependencies are installed is to install the `bioconda` package. Alternatively, DROP can be installed with `pip`. In this case the following dependencies must be met:

- Programming languages:
  - `python`  $\geq 3.6$  and `pip`  $\geq 19.1$
  - `R`  $\geq 3.6$ ,  $\leq 4.0.2$  and corresponding `bioconductor` version
- Commandline tools:
  - GNU `bc`



- GNU wget
- tabix
- samtools >= 1.9
- bcftools >= 1.9
- GATK >= 4.1.8
- graphviz
- pandoc

---

**Note:** If you are using an already existing R installation, make sure that the R and bioconductor versions match. Otherwise, use the newest versions of R and bioconductor.

---

At the first invocation, all the necessary R packages will be installed. As this is a lengthy process, it might be desirable to install them in advance.

```
# optional
Rscript <path/to/drop/repo>/drop/installRpackages.R drop/requirementsR.txt
```



---

## Preparing the Input Data

---

The input files of DROP are:

- BAM files from RNA-seq (and their respective index files)
- VCF files from either WES or WGS (and their respective index files). Only used for the MAE module
- a configuration file containing the different parameters
- a sample annotation file
- a gene annotation file (gtf)
- a reference genome file (fasta, and its respective index)

For more details see the Materials section of the [DROP manuscript](#).

### 2.1 Config file

The config file is in [YAML](#) format. It is composed of general and module-specific parameters. In [YAML](#), a variable can be of the following types: boolean, string, numeric, list and dictionary. They are declared by writing the variable name followed by a colon, a space, and the value, for example:

```
# A boolean is binary and can be true or false
boolean_var: true    # or false

# A string is a text. Quotation marks are not needed.
string_var: whatever text

# A numeric can be an integer or a real number. A dot separates a decimal.
numeric_var: 0.05

# A list is a collection of elements of the same type.
list_var:
  - element_1    # elements are indented
```

(continues on next page)

(continued from previous page)

```
- element_2

# A dictionary contains key-value pairs. It is a collection of multiple
# elements where the key is a string and the value any type.
dictionary_var:
  key_1: value_1
  key_2: value_2
```

Now we describe the different parameters needed in DROP. When providing a path to a file or directory, please provide the *full system path*.

### 2.1.1 Global parameters

These parameters are applied to multiple modules and as a result should be consistent throughout the data you are analyzing

Parameter	Type	Description	Default/Examples
project-Title	character	Title of the project to be displayed on the rendered HTML output	Project 1
htmlOutputPath	character	Full path of the folder where the HTML files are rendered	/data/project1/htmlOutput
indexWith-Folder-Name	boolean	if true, the basename of the project directory will be used as prefix for the index.html file	true
genome-Assembly	character	Either hg19/hs37d5 or hg38/GRCh38, depending on the genome assembly used for mapping	/data/project1
sampleAnnotation	character	Full path of the sample annotation table	/data/project1/sample_annotation.tsv
root	character	Full path of the folder where the sub-directories processed_data and processed_results will be created containing DROP's output files.	/data/project1
genome	character	Full path of a human reference genome fasta file	/path/to/hg19.fa
genome	dictionary	(Optional) Multiple fasta files can be specified when RNA-seq BAM files belong to different genome. assemblies (eg, ncbi, ucsc).	ncbi: /path/to/hg19_ncbi.fa ucsc: /path/to/hg19_ucsc.fa
geneAnnotation	dictionary	A key-value list of the annotation name (key) and the full path to the GTF file (value). More than one annotation file can be provided.	anno1: /path/to/gtf1.gtf anno2: /path/to/gtf2.gtf
hpoFile	character	Full path of the file containing HPO terms. If null (default), it reads it from our webserver. Refer to <a href="#">files-to-download</a>	/path/to/hpo_file.tsv
tools	dictionary	A key-value list of different commands (key) and the command (value) to run them	gatkCmd: gatk bcftoolsCmd: bcftools samtoolsCmd: samtools

### 2.1.2 Export counts dictionary

These parameters are directly used by the `exportCounts` snakemake command. This section is used to designate which aberrant expression and aberrant splicing groups should be exported into datasets that can be shared. To avoid sharing sensitive data, only the canonical annotations as described by *geneAnnotations* are exported. Only the groups excluded by *excludeGroups* are not exported.

Parameter	Type	Description	Default/Examples
geneAnnotations	list	key(s) from the <code>geneAnnotation</code> parameter, whose counts should be exported	- gencode34
excludeGroups	list	aberrant expression and aberrant splicing groups whose counts should not be exported. If <code>null</code> all groups are exported.	- group1

### 2.1.3 Aberrant expression dictionary

These parameters are directly used by the `aberrantExpression` snakemake command. Aberrant expression groups must have at least 10 samples per group. To use external counts please see the `Using External Counts` section.

Parameter	Type	Description	Default/Examples
run	boolean	If true, the module will be run. If false, it will be ignored.	true
groups	list	DROp groups that should be executed in this module. If not specified or <code>null</code> all groups are used.	- group1 - group2
minIds	numeric	A positive number indicating the minimum number of samples that a group needs in order to be analyzed. We recommend at least 50.	1
fpkmCutoff	numeric	A positive number indicating the minimum FPKM per gene that 5% of the samples should have. If a gene has less it is filtered out.	1 # suggested by OTRIDER
implementation	character	Either 'autoencoder', 'pca' or 'peer'. Methods to remove sample covariation in OTRIDER.	autoencoder
zScoreCutoff	numeric	A non-negative number. Z scores (in absolute value) greater than this cutoff are considered as outliers.	0
padjCutoff	numeric	A number between (0, 1] indicating the maximum FDR an event can have in order to be considered an outlier.	0.05
maxTested-Dimension-Proportion	numeric	An integer that controls the maximum value that the encoding dimension can take. Refer to <i>advanced-options</i> .	3
yieldSize	numeric	An integer that sets the batch size for counting reads within a bam file. If memory issues persist lower the <code>yieldSize</code> .	2000000
genesToTest	character	Full path to a yaml file specifying lists of candidate genes per sample to test during FDR correction. See the documentation for details on the structure of this file.	/path/to/ genes_to_test. yaml

### 2.1.4 Aberrant splicing dictionary

These parameters are directly used by the `aberrantSplicing` snakemake command. Each group must have at least 10 samples. This module uses FRASER to detect aberrant splicing. We recently developed an improved version of FRASER that uses the Intron Jaccard Index instead of percent spliced in and splicing efficiency to call aberrant splicing. To use this improved version, set the `FRASER_version` parameter to 'FRASER2'. When switching between FRASER versions, we recommend running DROp in a separate folder for each version. To use external counts, refer to the `Using External Counts` section.

Parameter	Type	Description	Default/Examples
run	boolean	If true, the module will be run. If false, it will be ignored.	true
groups	list	Same as in aberrant expression.	# see aberrant expression example
minIds	numeric	Same as in aberrant expression.	1
recount	boolean	If true, it forces samples to be recounted.	false
longRead	boolean	Set to true only if counting Nanopore or PacBio long reads.	false
keepNon-Standard-Chrs	boolean	Set to true if non standard chromosomes are to be kept for further analysis.	false
filter	boolean	If false, no filter is applied. We recommend filtering.	true
minExpressionInOneSample	numeric	The minimal read count in at least one sample required for an intron to pass the filter.	20
quantileMinExpression	numeric	The minimum total read count (N) an intron needs to have at the specified quantile across samples to pass the filter. See <code>quantileForFiltering</code> .	10
quantileForFiltering	numeric	Defines at which percentile the <code>quantileMinExpression</code> filter is applied. A value of 0.95 means that at least 5% of the samples need to have a total read count $N \geq \text{quantileMinExpression}$ to pass the filter.	0.95
min-DeltaPsi	numeric	The minimal variation (in delta psi) required for an intron to pass the filter.	0.05
implementation	character	Either 'PCA' or 'PCA-BB-Decoder'. Methods to remove sample covariation in FRASER.	PCA
FRASER_version	character	Either 'FRASER' or 'FRASER2'.	FRASER
deltaPsi-Cutoff	numeric	A non-negative number. Delta psi values greater than this cutoff are considered as outliers. Set to 0.1 when using FRASER2.	0.3 # suggested by FRASER
padjCutoff	numeric	Same as in aberrant expression.	0.1
max-TestedDimension-Proportion	numeric	Same as in aberrant expression.	6
genesToTest	character	Same as in aberrant expression.	/path/to/genes_to_test.yaml

### 2.1.5 Mono-allelic expression (MAE) dictionary

These parameters are directly used by the `mae` snakemake command. MAE groups are not bound by a minimum number of samples, but require additional information in the sample annotation table.

Parameter	Type	Description	Default/Examples
run	boolean	If true, the module will be run. If false, it will be ignored.	true
groups	list	Same as in aberrant expression.	# see aberrant expression example
gatkIgnoreHeaderCheck	boolean	If true (recommended), it ignores the header warnings of a VCF file when performing the allelic counts	true
padjCutoff	numeric	Same as in aberrant expression.	0.05
allelicRatioCutoff	numeric	A number between [0.5, 1) indicating the maximum allelic ratio $\text{allele1}/(\text{allele1}+\text{allele2})$ for the test to be significant.	0.8
addAF	boolean	Whether or not to add the allele frequencies from gnomAD	true
maxAF	numeric	Maximum allele frequency (of the minor allele) cut-off. Variants with AF equal or below this number are considered rare.	0.001
maxVarFreqCohort	numeric	Maximum variant frequency among the cohort.	0.05
qcVcf	character	Full path to the vcf file used for VCF-BAM matching. Refer to <a href="#">files-to-download</a> .	/path/to/qc_vcf.vcf.gz
qcGroups	list	Same as “groups”, but for the VCF-BAM matching	# see aberrant expression example
dnaRnaMatchCutoff	numeric	fraction (0-1) used to separate “matching” samples and “non-matching” samples comparing the DNA and RNA data during QC	0.85

### 2.1.6 RNA Variant Calling dictionary

Calling variants on RNA-seq data may be useful for researchers who do not have access to variant calls from genomic data. While variant calling from WES and WGS technologies may be more traditional (and reliable), variant calling from RNA-Seq data can provide additional evidence for the underlying causes of aberrant expression or splicing. The RNA variant calling process uses information from multiple samples (as designated by the `groups` variable) to improve the quality of the called variants. However, the larger the group size, the more costly the computation is in terms of time and resources. To prioritize accuracy, include many samples in each `DROP_GROUP`, and to prioritize speed up computation, separate samples into many groups. Additionally, certain vcf and bed files must be included to further boost the quality of the called variants (refer to [files-to-download](#)).



Parameter	Type	Description Default/Examples	
run	boolean	If true, the module will be run. If false, it will be ignored.	true
groups	list	Same as in aberrant expression.	# see aberrant expression example
high-QualityVCFs	list	File paths where each item is the path to a vcf file. Each vcf file describes known high quality variants, which are used to recalibrate sequencing scores. Refer to <i>files-to-download</i>	- known_indels.vcf - known_SNPs.vcf
dbSNP	character	Location of the dbSNP .vcf file. This improves both recalibrating sequencing scores, as well as variant calling precision. Refer to <i>files-to-download</i>	path/to/dbSNP.vcf
repeat_mask	character	Location of the RepeatMask .bed file. Refer to <i>files-to-download</i>	path/to/RepeatMask.bed
createSingleVCF	boolean	If true, splits the multi-sample VCF file into individual sample VCF files. This only subsets the larger vcf sample.	true
addAF	boolean	Whether or not to add the allele frequencies from gnomAD	true
maxAF	numeric	Maximum allele frequency (of the minor allele) cut-off. Variants with AF equal or below this number are considered rare.	0.001
max-VarFreqCohort	numeric	Maximum variant frequency among the cohort.	0.05
minAlt	numeric	Integer describing the minimum required reads that support the alternative allele. We recommend a minimum of 3 if further filtering on your own. 10 otherwise.	3
hcArgs	character	String describing additional arguments for GATK haplocaller. Refer to <i>advanced-options</i> .	" "
yield-Size	numeric	An integer that sets the batch size for counting reads within a vcf file. If memory issues persist during <code>batch_data_table</code> lower the yield-Size.	100000

## 2.2 Modularization of DROP

DROP allows to control which modules to run via the `run` variable in the config file. By default, each module is set to `run: true`. Setting this value to `false` stops a particular module from being run. This will be noted as a warning at the beginning of the `snakemake` run, and the corresponding module will be renamed in the `Scripts/` directory.

For example, if the `AberrantExpression` module is set to `false`, the `Scripts/AberrantExpression/` directory will be renamed to `Scripts/_AberrantExpression/` which tells DROP not to execute this module.

## 2.3 Creating the sample annotation table

For a detailed explanation of the columns of the sample annotation, please refer to Box 3 of the [DROPP manuscript](#). Although some information has been updated since publication, please use this documentation as the preferred syntax/formatting.

Each row of the sample annotation table corresponds to a unique pair of RNA and DNA samples derived from the same individual. An RNA assay can belong to one or more DNA assays, and vice-versa. If so, they must be specified in different rows. The required columns are `RNA_ID`, `RNA_BAM_FILE` and `DROP_GROUP`, plus other module-specific ones (see [DROPP manuscript](#)).

The following columns describe the RNA-seq experimental setup: `PAIRED_END`, `STRAND`, `COUNT_MODE` and `COUNT_OVERLAPS`. They affect the counting procedures of the aberrant expression and splicing modules. For a detailed explanation, refer to the documentation of [HTSeq](#).

To run the MAE module, the columns `DNA_ID` and `DNA_VCF_FILE` are needed. MAE can not be run in samples using external counts as we need to use the `RNA_BAM_FILE` to count reads supporting each allele of the heterozygous variants found in the `DNA_VCF_FILE`.

In case RNA-seq BAM files belong to different genome assemblies (eg, ncbi, ucsc), multiple reference genome fasta files can be specified. Add a column called `GENOME` that contains, for each sample, the key from the `genome` parameter in the config file that matches its genome assembly (eg, ncbi or ucsc).

The sample annotation file must be saved in the tab-separated values (tsv) format. The column order does not matter. Also, it does not matter where it is stored, as the path is specified in the config file. Here we provide some examples on how to deal with certain situations. For simplicity, we do not include all possible columns in the examples.

### 2.3.1 Using External Counts

DROPP can utilize external counts for the `aberrantExpression` and `aberrantSplicing` modules which can enhance the statistical power of these modules by providing more samples from which we can build a distribution of counts and detect outliers. However this process introduces some particular issues that need to be addressed to make sure it is a valuable addition to the experiment.

In case external counts are included, add a new row for each sample from those files (or a subset if not all samples are needed). Add the columns: `GENE_COUNTS_FILE` (for aberrant expression), `GENE_ANNOTATION`, and `SPLICE_COUNTS_DIR` (for aberrant splicing). These columns should remain empty for samples processed locally (from `RNA_BAM`).

#### Aberrant Expression

Using external counts for aberrant expression forces you to use the exact same gene annotation for each external sample as well as using the same gene annotation file specified in the config file `Global parameters` section. This is to avoid potential mismatching on counting, 2 different gene annotations could drastically affect which reads are counted in which region drastically skewing the results.

The user must also use special consideration when building the sample annotation table. Samples using external counts need only `RNA_ID` which must exactly match the column header in the external count file `DROP_GROUP`, `GENE_COUNTS_FILE`, and `GENE_ANNOTATION` which must contain the exact key specified in the config. The other columns should remain empty.

Using `exportCounts` generates the sharable `GENE_COUNTS_FILE` file in the appropriate `ROOT_DIR/Output/processed_results/exported_counts/ sub-directory`.

## Aberrant Splicing

Using external counts for aberrant splicing reduces the number of introns processed to only those that are exactly the same between the local and external junctions. Because rare junctions may be personally identifiable the `exportCounts` command only exports regions canonically mentioned in the `gtf` file. As a result, when merging the external counts with the local counts we only match introns that are **exact** between the 2 sets, this is to ensure that if a region is missing we don't introduce 0 counts into the distribution calculations.

The user must also use special consideration when building the sample annotation table. Samples using external counts need only `RNA_ID` which must exactly match the column header in the external count file `DROP_GROUP`, and `SPLICE_COUNTS_DIR`. `SPLICE_COUNTS_DIR` is the directory containing the set of 5 needed count files. The other columns should remain empty.

Using `exportCounts` generates the necessary files in the appropriate `ROOT_DIR/Output/processed_results/exported_counts/` sub-directory

`SPLICE_COUNTS_DIR` should contain the following:

- `k_j_counts.tsv.gz`
- `k_theta_counts.tsv.gz`
- `n_psi3_counts.tsv.gz`
- `n_psi5_counts.tsv.gz`
- `n_theta_counts.tsv.gz`

## Publicly available DROP external counts

You can find different sets of publicly available external counts to add to your analysis on our [github page](#)

If you want to contribute with your own count matrices, please contact us: [yepez at in.tum.de](mailto:yepez@in.tum.de).

### 2.3.2 External count examples

In case counts from external matrices are to be integrated into the analysis, the sample annotation must be built in a particular way A new row must be added for each sample from the count matrix that should be included in the analysis. The `RNA_ID` must match the column header of the external files, the `RNA_BAM_FILE` must not be specified. The `DROP_GROUP` of the local and external samples that are to be analyzed together must be the same. For aberrant expression, the `GENE_ANNOTATION` of the external counts and the key of the `geneAnnotation` parameter from the config file must match.

This example will use the `DROP_GROUP BLOOD_AE` for the aberrant expression module (containing S10R, EXT-1R, EXT-2R) and the `DROP_GROUP BLOOD_AS` for the aberrant expression module (containing S10R, EXT-2R, EXT-3R)

RNA_ID	DNA_ID	DROP_GROUP	RNA_BAM_FILE	GENE_COUNTS_FILE	GENE_ANNOTATION	SPLICE_COUNTS_DIR
S10R	S10G	BLOOD_AE,BLOOD_AS	/path/to/S10R.BAM			
EXT-1R		BLOOD_AE		/path/to/externalCounts	gene34	
EXT-2R		BLOOD_AE,BLOOD_AS		/path/to/externalCounts	gene34	/path/to/externalCountDir
EXT-3R		BLOOD_AS				/path/to/externalCountDir

## 2.4 Limiting FDR correction to subsets of genes of interest

In addition to returning transcriptome-wide results, DROP provides the option to limit the FDR correction to user-provided genes of interest in the `aberrantExpression` and `aberrantSplicing` modules. These could e.g. be all OMIM genes. It is also possible to provide sample-specific genes such as all genes with a rare splice region variant for each sample. To use this feature, a YAML file containing the set(s) of genes to test (per sample or for all samples) needs to be specified in the `genesToTest` parameter of the `aberrantExpression` and `aberrantSplicing` modules in the config file. If no file is provided, only transcriptome-wide results will be reported. Otherwise, the results tables of the `aberrantExpression` and `aberrantSplicing` modules will additionally report aberrant events passing the cutoffs based on calculating the FDR with respect to the genes in the provided lists.

### 2.4.1 Creating the YAML file specifying subsets of genes to test

The file containing the list of genes (HGNC symbols) to be tested must be a YAML file, where the variable names specify the name of each set of tested genes. In the output of DROP, this name will be used to identify the set in the results table. Each set can either be a list of genes, in which case the set will be tested for all samples. Alternatively (and additionally), sample-specific sets can be created by giving the `RNA_ID` of the sample for which the set should be used as the name (see example below). This YAML file can be created in R using `yaml::write_yaml(subsetList, filepath)`, where `subsetList` is a named list of named lists containing the sets of genes to test. In the following example, the name of the global set of genes is `Genes_to_test_on_all_samples` and the name of the sample-specific set is `Genes_with_rare_splice_variants`:

Example content of `/path/to/genes_to_test.yaml`:

```
Genes_to_test_on_all_samples:
- BTG3
- GATD3B
- PKNOX1
- APP
- RRP1
- WRB-SH3BGR
- SLC19A1
Genes_with_rare_splice_variants:
  sample1:
- ABCG1
- MCOLN1
- SLC45A1
  sample2:
- CLIC6
- ATP5PO
- WRB
- ETS2
- HLCS
```

## 2.5 Files to download

The following files can be downloaded from our [public repository](#).

1. VCF file containing different positions to be used to match DNA with RNA files. The file name is `qc_vcf_1000G_{genome_build}.vcf.gz`. One file is available for each genome build (hg19/hs37d5 and hg38/GRCh38). Download it together with the corresponding `.tbi` file. Indicate the full path to the vcf file in the `qcVcf` key in the mono-allelic expression dictionary. This file is only needed for the MAE module. Otherwise, write `null` in the `qcVcf` key.

2. Text file containing the relations between genes and phenotypes encoded as HPO terms. The file name is `hpo_genes.tsv.gz`. Download it and indicate the full path to it in the `hpoFile` key. The file is only needed in case HPO terms are specified in the sample annotation. Otherwise, write `null` in the `hpoFile` key.

3. For the `rnaVariantCalling` module known high quality variants are needed to calibrate variant and sequencing scores to be used in the `rnaVariantCalling` module in the `highQualityVCF` config parameter. These and the associated `.tbi` indexes can be downloaded for hg19 at our [public repository](#) and for hg38 through the Broad Institute's [resource bundle](#).

hg19

- `Mills_and_1000G_gold_standard.indels.hg19.sites.chrPrefix.vcf.gz`
- `1000G_phase1.snps.high_confidence.hg19.sites.chrPrefix.vcf.gz`

hg38

- `Mills_and_1000G_gold_standard.indels.hg38.vcf.gz`
- `Homo_sapiens_assembly38.known_indels.vcf.gz`

We also recommend using the variants from dbSNP which is quite large. You can download them and their associated `.tbi` indexes from [NCBI](#)

- follow links for the current version (`human_9606/VCF/00-All.vcf.gz`) or older assemblies (eg. `human_9606_b151_GRCh37p13/VCF/00-All.vcf.gz`)

The repeat masker file is used to filter hard-to-call regions. In general, this removes false-positive calls, however, some targeted and known splicing defects lie within these repeat regions. Understand that this filter is labeled `Mask` in the result VCF files. You can download the repeat mask and associated `.idx` on our [public repository](#). for the `repeat_mask` config parameter.

## 2.6 Example of RNA replicates

RNA_ID	DNA_ID	DROP_GROUP	RNA_BAM_FILE	DNA_VCF_FILE
S10R_B	S10G	BLOOD	/path/to/S10R_B.BAM	/path/to/S10G.vcf.gz
S10R_M	S10G	MUSCLE	/path/to/S10R_M.BAM	/path/to/S10G.vcf.gz

### 2.6.1 Example of DNA replicates

RNA_ID	DNA_ID	DROP_GROUP	RNA_BAM_FILE	DNA_VCF_FILE
S20R	S20E	WES	/path/to/S20R.BAM	/path/to/S20E.vcf.gz
S20R	S20G	WGS	/path/to/S20R.BAM	/path/to/S20G.vcf.gz

### 2.6.2 Example of a multi-sample vcf file

RNA_ID	DNA_ID	DROP_GROUP	RNA_BAM_FILE	DNA_VCF_FILE
S10R	S10G	WGS	/path/to/S10R.BAM	/path/to/multi_sample.vcf.gz
S20R	S20G	WGS	/path/to/S20R.BAM	/path/to/multi_sample.vcf.gz

---

## 2.7 Advanced options

A local copy of DROPP can be edited and modified. For example, the user might want to add new plots to the Summary scripts, add additional columns to the results tables, or modify the number of threads allowed for a script.

---

**Note:** DROPP needs to be installed from a local directory using `pip install -e <path/to/drop-repo>` so that any changes in the code will be available in the next pipeline run. Any changes made to the R code need to be updated with `drop update` in the project directory.

---

The aberrant expression and splicing modules use a denoising autoencoder to correct for sample covariation. This process reduces the fitting space to a dimension smaller than the number of samples  $N$ . The encoding dimension is optimized. By default, the maximum value in the search space is  $N/3$  for the aberrant expression, and  $N/6$  for the aberrant splicing case. The user can specify the denominator with the parameter `maxTestedDimensionProportion`.

DROPP allows that BAM files from RNA-seq from samples belonging to the same *DROPP\_GROUP* were aligned to different genome assemblies from the same build (e.g., some to ucsc and others to ncbi, but all to either hg19 or hg38). If so, for the aberrant expression and splicing modules, no special configuration is needed. For the MAE and `rnaVariantCalling` modules, the different fasta files must be specified as a dictionary in the *genome* parameter of the config file, and, for each sample, the corresponding key of the *genome* dictionary must be specified in the *GENOME* column of the sample annotation. In addition, DROPP allows that BAM files from RNA-seq were aligned to one genome assembly (eg ucsc) and the corresponding VCF files from DNA sequencing to another genome assembly (eg ncbi). If so, the assembly of the reference genome fasta file must correspond to the one of the BAM file from RNA-seq.

Specific haplotype parameters can be denoted in the config file to further customize the RNA-seq variant calling. The different available parameters can be found in the [HaplotypeCaller GATK documentation](#). One example for the value in the config file would be “`-assembly-region-padding 100 -base-quality-score-threshold 18`”.

DROP is a `Snakemake` pipeline, so it is called with the `snakemake` command.

### 3.1 Dry run

Open a terminal in your project repository. Execute

```
snakemake --cores 1 -n
```

This will perform a *dry-run*, which means it will display all the steps (or rules) that need to be executed. To also display the reason why those rules need to be executed, run

```
snakemake --cores 1 -nr
```

A simplified dry-run can be achieved using the `-q` parameter.

```
snakemake --cores 1 -nq
```

Calling `snakemake --cores 1` without any additional parameters will execute the whole workflow. `Snakemake` requires you to designate the number of cores.

### 3.2 Parallelizing jobs

DROP's steps are computationally heavy, therefore it is highly recommended to run them in parallel. `Snakemake` automatically determines the steps that can be parallelized. The user simply needs to specify the maximum number of cores that `Snakemake` can take, e.g. for 10 cores:

```
snakemake --cores 10
```

### 3.3 Executing subworkflows

Every module can be called independently.

```
snakemake <subworkflow>
```

Subworkflow	Description
aberrantExpression	Aberrant expression pipeline
aberrantSplicing	Aberrant splicing pipeline
mae	Monoallelic expression pipeline
sampleQC	DNA-RNA matching (already part of mae, but can be executed independently as well)
rnaVariantCalling	RNA Variant Calling pipeline

For example, to run the aberrant expression pipeline with 10 cores, execute the following

```
snakemake aberrantExpression --cores 10
```

### 3.4 Rerunning the pipeline

When DROP is updated or jobs fail, the following commands can be used to rerun and troubleshoot.

#### 3.4.1 Unlocking the pipeline

While running, Snakemake *locks* the directory. If, for a whatever reason, the pipeline was interrupted, the directory might be kept locked. If this is the case, call

```
snakemake unlock
```

#### 3.4.2 Updating DROP

The developers of DROP are active in making DROP a better tool. As a result there are often bug fixes or improvements that are implemented and released in new versions. You can check them out in the *What's new* section of the [README](#).

When updating DROP we recommend using the conda/mamba functions to maintain any dependencies that could be related.

```
mamba update drop
```

If you were working with a pip installation of DROP then you would need to reinstall using pip directly from github.

```
pip install git+https://github.com/gagneurlab/drop.git
```

Once you have successfully bumped the DROP version to the latest, you will still need to update your project folder. *drop update* will reset the local project's *Scripts/* directory to match the installed version, so be sure to save any additional scripts or analyses in another location.

To complete your update, you must run the following to get your local directory to match the version:

```
drop update
```



### 3.4.3 Skipping recomputation of files

If the pipeline is interrupted and restarted, it will continue with the last unsuccessful job in the job graph. If a script is updated with minor change, e.g. when calling `drop update`, all the jobs of the modified script and its downstream steps will be rerun. However, in some cases one might want to keep the intermediate files instead and continue with the missing files. In order to do so, first execute

```
snakemake <rule> --touch
```

for whichever rule or module you want to continue the computation. The `--touch` command touches all output files required by the pipeline that have already been computed. Omitting the rule will lead to accessing the complete pipeline. Afterwards, run

```
snakemake unlock
```

Overall, we recommend reading the snakemake documentation for further fine-tuning of the execution.



---

## Results and Output of DROP

---

DROP is intended to help researchers use RNA-Seq data in order to detect genes with aberrant expression, aberrant splicing and mono-allelic expression. By simplifying the workflow process we hope to provide easy-to-read HTML files and output files. This section explains the results files. The paths of the output files correspond to the ones from the demo (that can be run with the following code snippet):

```
#install drop
mamba create -n drop_env -c conda-forge -c bioconda drop
conda activate drop_env

mkdir drop_demo
cd drop_demo
drop demo

snakemake -c1
```

### 4.1 Aberrant Expression

#### 4.1.1 HTML file

Looking at the resulting `Output/html/drop_demo_index.html` we can see the `AberrantExpression` tab at the top of the screen. The Overview tab contains links to the:

- **Counts Summaries for each aberrant expression group**
  - number of local and external samples
  - Mapped reads and size factors for each sample
  - histograms showing the mean count distribution with different conditions
  - expressed genes within each sample and as a dataset
- **Outsider Summaries for each aberrant expression group**

- aberrantly expressed genes per sample
- correlation between samples before and after the autoencoder
- biological coefficient of variation
- aberrant samples
- results table
- **Files for each aberrant expression group**
  - **OUTRIDER datasets**
    - \* Follow the [OUTRIDER vignette](#) for individual OUTRIDER object file (ods) analysis.
  - **Results tables**
    - \* `results.tsv` this text file contains only the significant genes and samples that meet the cutoffs defined in the config file for `padjCutoff` and `zScoreCutoff`.

### 4.1.2 Local result files

Additionally the `aberrantExpression` module creates the file `Output/processed_results/aberrant_expression/{annotation}/outrider/{drop_group}/OUTRIDER_results_all.Rds`. This file contains the entire OUTRIDER results table regardless of significance.

## 4.2 Aberrant Splicing

### 4.2.1 HTML file

Looking at the resulting `Output/html/drop_demo_index.html` we can see the `AberrantSplicing` tab at the top of the screen. The Overview tab contains links to the:

- **Counting Summaries for each aberrant splicing group**
  - number of local and external samples
  - number introns/splice sites before and after merging
  - comparison of local and external mean counts
  - histograms showing the junction expression before and after filtering and variability
- **FRASER Summaries for each aberrant splicing group**
  - the number of samples, introns, and splice sites
  - correlation between samples before and after the autoencoder
  - results table
- **Files for each aberrant splicing group**
  - **FRASER datasets (fds)**
    - \* Follow the [FRASER vignette](#) for individual FRASER object file (fds) analysis.
  - **Results tables**
    - \* `results_per_junction.tsv` this text file contains only significant junctions that meet the cutoffs defined in the config file.

## 4.2.2 Local result files

Additionally the `aberrantSplicing` module creates the following file `Output/processed_results/aberrant_splicing/results/{annotation}/fraser/{drop_group}/results.tsv`. This text file contains only significant junctions that meet the cutoffs defined in the config file, aggregated at the gene level. Any sample/gene pair is represented by only the most significant junction.

## 4.3 Mono-allelic Expression

### 4.3.1 HTML file

Looking at the resulting `Output/html/drop_demo_index.html` we can see the `MonoallelicExpression` tab at the top of the screen. The `Overview` tab contains links to the:

- **Results for each mae group**
  - number of samples, genes, and mono-allelically expressed heterozygous SNVs
  - a cascade plot that shows additional filters
  - histogram of inner cohort frequency
  - summary of the cascade plot and results table
- **Files for each mae group**
  - **Allelic counts**
    - \* a directory containing the allelic counts of heterozygous variants
  - **Results data tables of each sample (.Rds)**
    - \* Rds objects containing the full results table regardless of MAE status
  - **Significant MAE results tables**
    - \* a link to the results file
    - \* Only contains significant MAE for the alternative allele results and results that pass the config file cutoffs
- **Quality Control**
  - **QC Overview**
    - \* For each mae group QC checks for DNA/RNA matching

### 4.3.2 Local result files

Additionally the `mae` module creates the following files:

- `Output/processed_results/mae/{drop_group}/MAE_results_all_{annotation}.tsv.gz`
  - this file contains the MAE results of all heterozygous SNVs regardless of significance
- `Output/processed_results/mae/{drop_group}/MAE_results_{annotation}.tsv`
  - this is the file linked in the HTML document and described above
- `Output/processed_results/mae/{drop_group}/MAE_results_{annotation}_rare.tsv`

- this file is a subset of `MAE_results_{annotation}.tsv` with only the variants that pass the allele frequency cutoffs. If `add_AF` is set to `true` in config file must meet minimum AF set by `max_AF`. Additionally, the inner-cohort frequency must meet the `maxVarFreqCohort` cutoff

## 4.4 RNA Variant Calling

### 4.4.1 HTML file

Looking at the resulting `Output/html/drop_demo_index.html` we can see the `rnaVariantCalling` tab at the top of the screen. The Overview tab contains links to the:

- **Results for each rvc batch**
  - a table summarizing the variants and genotypes that pass the variant calling filters for each sample
    - \* `FILTER`: explained below
    - \* `cohortFreq`: frequency of the variant within the batch (number of samples with the variant / total samples)
    - \* `MAX_AF`: frequency of the variant from **gnomAD** if enabled
  - a subset table showing only the `PASS_rare` variants
- **Boxplot and underlying table showing the distribution of variants and the effect of various filters, split by genotype**
  - **The following labels are applied and are not excluded from the final `.vcf` files.**
    - \* `PASS_common`: passes variant calling thresholds and fails either `max_AF` or `maxVarFreqCohort` cutoffs
    - \* `PASS_rare`: passes variant calling thresholds and config `max_AF` and `maxVarFreqCohort` cutoffs
    - \* `Seq_filter`: fails one of the default variant calling filters
    - \* `Mask`: variant falls in a repeat/mask region
    - \* `minALT`: variant passes `Seq_filter` but doesn't meet config `minALT` criteria
- Boxplot and underlying table showing the number of variants that pass or fail the different filters

### 4.4.2 Local result files

Additionally the `rnaVariantCalling` module creates the following output directories:

- **Output/processed\_results/rnaVariantCalling/batch\_vcfs**
  - this directory contains the multi-sample vcf files for each batch
- **Output/processed\_results/rnaVariantCalling/sample\_vcfs**
  - this directory contains the single-sample vcf files if the config parameter `createSingleVCF` is set to `true`
- **Output/processed\_results/rnaVariantCalling/data\_tables**
  - this directory contains data tables for each batch of vcfs

### MIT License

Copyright (c) 2019, Michaela Mueller, Vicente Yopez, Christian Mertes, Daniela Andrade, Julien Gagneur

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.





## CHAPTER 6

---

### Quickstart

---

DROP is available on [bioconda](#). We recommend using a dedicated conda environment. (installation time: ~ 10min)  
Use *mamba* instead of *conda* as it provides more reliable and faster dependency solving.

```
mamba create -n drop -c conda-forge -c bioconda drop
```

Test installation with demo project

```
mkdir ~/drop_demo  
cd ~/drop_demo  
drop demo
```

The pipeline can be run using [snakemake](#) commands

```
snakemake --cores 1 -n # dryrun  
snakemake --cores 1
```

Expected runtime: 25 min

For more information on different installation options, refer to *Installation*.