GVAF (Germline variation annotation & filtering)

2018.02.12

Introduction

GVAF (called Germline Variant Annotation Filtering) is command-line software that is designed to provide a flexible filtering function and annotation for TXT files (called annotated files) consisting of arbitrary fields such as the Annovar output-like file format to explore genetic variants.

Overview

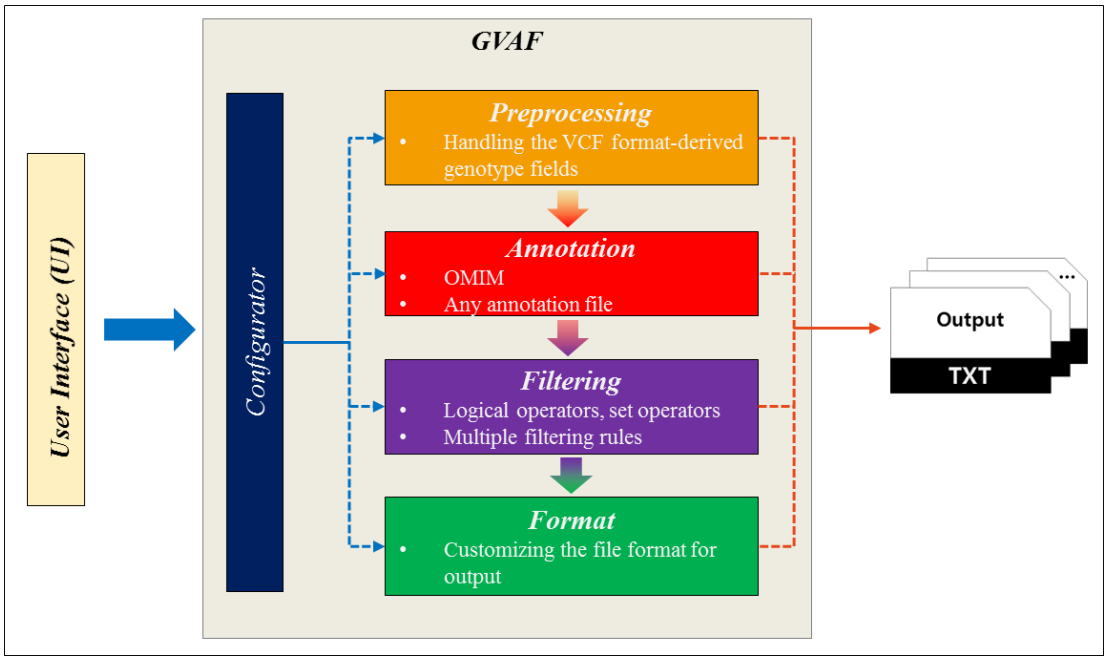


Figure 1. The architecture of GVAF.

Figure 1 shows the software architecture of GVAF that provides the various functions for genetic variant exploration. GVAF is composed of 5 modules and each module provides the different function for exploring the variants. You can execute GVAF program using user Interface (UI) tool with the options. UI can parse the user-supplied options and pass them to the configurator module of GVAF. According to the user-supplied option, the configurator module in GVAF manages the environmental settings and generates the execution flow. The remaining modules (e.g., preprocessing, annotation, filtering, format) provide the function to explore the interesting or relevant variants. GVAF can perform the following main functions using several modules for handling genetic variants on the annotated files:

- Preprocessing: This function is used to convert from the VCF derived genotype data to user-friendly genotype-related fields.
- Annotation: This function is used to add the fields including further information about the variant.
- Filtering: This function is used to filter out variants using simple but flexible filtering expression.
- Format: This function is used to generate customized output files.

In additional, GVAF provides the function to handle the input files such as ANNOVAR-like output file including the field names with the null values. The field names with the null values in the input files are assigned in sequence using the letter (extra1~n).

GVAF can execute several modules together at the same time. Five modules (i.e., preprocessing, annotation, filtering, format) in GVAF for exploring genetic variants can be executed in sequential order according to each module priority. We prioritized each module in order of preprocessing, annotation, filtering, and format (see figure 1). The execution flow is generated based on the predefined priority and each module belong to the execution flow uses the result files of the previously executed module as input files except in initial module.

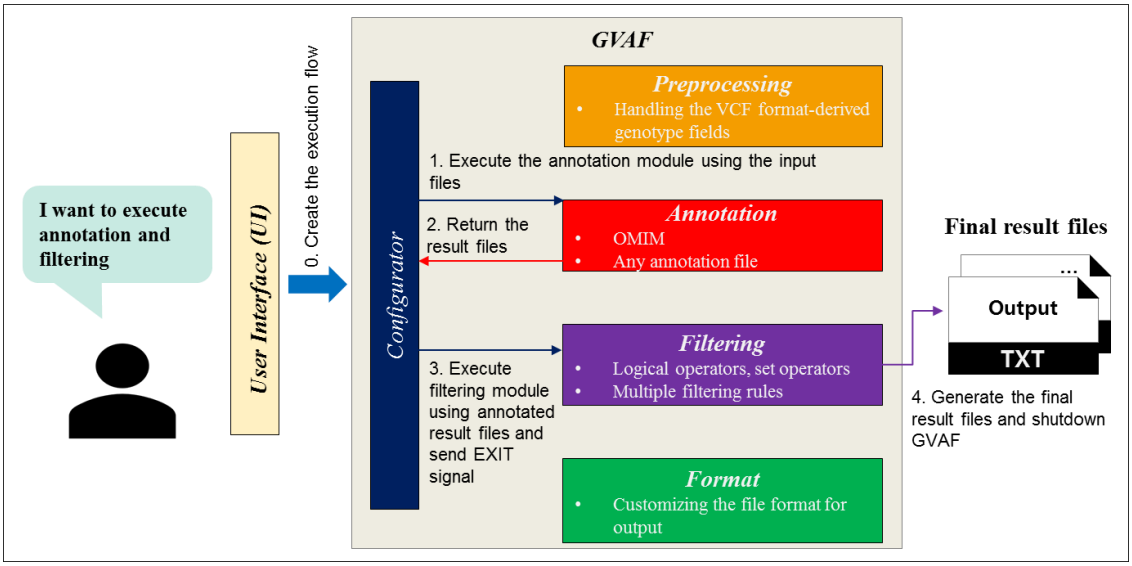


Figure 2. The execution step using annotation and filtering modules.

We demonstrate the execution step of GVAF when you want to execute the annotation module and the filtering module together in figure 2. Using UI, you can pass a command including the options related to annotation and filtering module and the input files into GVAF. The final result files can be generated according to the following procedure.

0. Create the execution flow according to the predefined module priority from a user-supplied command by the configurator module.
1. Execute the annotation module with the high priority using the input files.
2. Return the annotated result files to the configurator module.
3. Execute the filtering module using the result files of the annotation module as input and sent EXIT signal.
   - EXIT signal: This signal means to shutdown GVAF.
4. Generate the final result files and shutdown GVAF.

All result files of executed module are stored in individual directory named each module name of output directory and you can also get the intermediate result files of each module.

Getting started with GVAF

System requirements

Java version 1.7 (or higher)

Installing GVAF (for Linux system)

1. Download a file form the link.

```
# Go to home directory
cd

# Download GVAF
wget -O GVAF.zip "https://drive.google.com/uc?export=download&id=1-9-2Qs8e-Tq2MTNYkxkT5SWBqcu6q2xe"
```

2. Decompress the downloaded file into a work directory.

```
# Unzip downloaded file <downloaded file path>
unzip GVAF.zip
```

3. Set configuration on your computer.

```
# Go to GVAF directory
cd GVAF

# Copy GVAF direcory using "pwd" command
pwd
/home/tools/GVAF

# Append the following lines into "~/.bash_profile"
export GVAF_HOME="/home/tools/GVAF"
export PATH=$GVAF_HOME/scripts:$PATH

# Update "~/.bash_profile"
source ~/.bash_profile
```

4. Change permission of UI

```
# Go to scripts directory
cd scripts

# Grant executable permission for UI
chmod 755 *
```

User Interface (UI)

We provide two user interface tools (written Bash).

- gvaf: The script is used to execute the GVAF using the main functions (e.g., annotation, filtering, extracting specific fields etc.).
- gvaf_creation_annotation_file: This script generates a file that includes a specific column with no duplication for executing the annotation step on GVAF.

gvaf

The gvaf tool can be used with the command-line options and the options can be classified to five categories (i.e., basic, preprocessing, annotation, filtering, format).

Options

You can find the usage of the option in each category in Table 1. You can get a detailed description of each categories in page 6-16. The options marked with "[]" will automatically be set to default values or null value unless otherwise specified, and you can skip these options optionally.

Categories	Option	Description
Basic	-input <files or directory>	This option is used to set the input files using the file or directory paths. ➢ e.g., you want to set a single file (named test1.txt) as input. -input test1.txt ➢ e.g., you want to set multiple files using directories (named text_dir1, text2_dir2) as input. -input "text_dir1, text_dir2"
	[-outputdir] <directory>	This option is used to set output directory for storing the result files. If output directory configurated by the user does not exist, GVAF automatically creates a new directory. If you do not set this option, GVAF automatically creates a new directory (named "GVAF_reuslt_<date>") by default, and sets it as output directory path. ➢ e.g., you want to store the result files at /home/user/gvaf_result -outputdir /home/user/gvaf_result
	[-threads] <number>	This option is used to set the number of threads for parallel work. If you do not set this option, GVAF automatically sets 1 by default (i.e., non-parallel work). ➢ e.g., you want to execute parallel tasks using 5 threads. -threads 5
Preprocessing	-genotype_format_format <field name>	This option is used to set the field including the VCF format-derived genotype data format. ➢ e.g., you want to annotate genotype-

		related fields by referencing the value of field (named extra4) that is the VCF format-derived genotype format. -genotype_format_format extra4
	-genotype_field <field name>	This option is used to set the field including the VCF format-derived genotype data. ➢ e.g., you want to annotate genotype-related fields by referencing the value of field (named extra5) that is the VCF format-derived genotype data. -genotype_format_format extra5
Annotation	-annotate <annotation expression>	This option is used to specify the annotation expression. ➢ e.g., you want to add a certain line of an annotation file if the value of Gene from input files is equal to the value of OMIM.Gene field. -annotate “Gene==OMIM.Gene”
	-annotation_file <annotation file path or “omim”>	This option is used to set an annotation file including the lines consisting of annotated fields. You can use OMIM gene annotation file without an annotation file using “omim”. ➢ e.g., you want to execute OMIM gene annotation. -annotate_file “omim” ➢ e.g., you want to execute customized annotation using a file (annotation_file.txt) -annotate_file annotation_file.txt
	[-delimiter <character>]	This option is used to specify the delimiter for the input file field specified in the annotation expression including the multiple values. ➢ e.g., you want to extract a single value from the input file field that is multiple valued field with “value1, value2, …”. -delimiter “,”
Filtering	-FilteringRule_File <filteringrule file>	This option is used to set a file path (called FilteringRule file) specified in the filtering rule. You can find how to generate a FilteringRule file (see page 7). ➢ e.g., you want to set FilteringRule file (named FR.txt) -FilteringRule_File FilteringRule.txt
	-FieldDefinition_File <fielddefinition file>	This option is used to set a file path (called FieldDefinition file) including the information of the fields defined in a FilteringRule file. You can find a detailed description of FieldDefinition file (see page 14). ➢ e.g., you want to FieldDefinition file

		(named FD.txt) FieldDefinition_File FD.txt
	[-ListFiltering_field <field name>]	This option is used to set the field of the input files or result files for applying ListFiltering. You can find a detailed description of the ListFiltering function (see page 15). ➢ e.g., you want to use the field (named gene) for applying ListFiltering. -ListFiltering_field gene
	[-ListFiltering_file <ListFiltering file>]	This option is used to set a file (called ListFiltering file) including a list of values for executing ListFiltering. You can get how to create a ListFiltering file (see page 15). ➢ e.g., you want to execute ListFiltering using a file (named LF.txt). -ListFiltering_file LF.txt
	[-NAStrings <strings>]	This option is used to set the strings that represents the missing values. If each field in the input files has a different string that represents missing value, you can use "," (comma) for specifying the multiple missing values. ➢ e.g., the input files have "NA" of "NONE" as missing value. -NAStrings "NA,NONE"
Format	-format_file <format file>	This option is used to set a file path (called format file) including output file format. You can find a detailed description of format file (see page 15). ➢ e.g., you want to set a format file path (named format.txt) -format_file format.txt

Table 1. The overview of available options.

Basic

To perform program, GVAF requires the input files with the output location or the number of threads. You can set the output location and the number of threads selectively. All result files are stored in default output location if the output location is not set. GVAF are executed in nonparallel by default. But, GVAF supports the parallel works using the multiple-threading technique that is used to use the resource efficiently on heterogeneous environment and reduce the execution time. You can execute GVAF program in parallel using the number of threads. This category includes the options related to environmental settings for running GVAF program (e.g., the number of threads, input files, etc.).

Preprocessing

The genotype-related fields in VCF files is hard to apply the further functions such as annotation and filtering. To solve this problem, we provide the preprocessing functions for handling the VCF

derived genotype fields. This category includes the options related to the preprocessing function that is used to converts from the VCF format-derived genotype fields to the genotype-related fields (see table 2).

Printed field name	Description
het/hom	het/hom indicates heterozygous or homozygous based on GT (i.e., Genotype)
Totalread	totalread indicates total read based on AD (i.e., Allelic depths for the ref and alt alleles in the order listed)
Refread	refread indicates reference read based on AD (i.e., Allelic depths for the ref and alt alleles in the order listed)
Altread	altread indicates alternative read based on DP (i.e., Approximate read depth)
Genotypequality	Genotypequality indicates genotype quality based on GT (i.e., genotype quality)

Table 2. Conversion of genotype-related fields by GVAF.

Table 2 shows a detailed description of the genotype-related fields. Using the options in this category, you can automatically annotate the genotype-related fields (i.e., het/hom, totalread, refread, altread, genotypequality) based on VCF format-derived genotype fields of the input files.

Annotation

This category includes the options related to the annotation function. You can add the additional fields of an annotation file into the input files using the annotation function. GVAF provides the OMIM gene annotation file by default, and any annotation file can be used. You can execute the customized annotation task using the simple annotation expressions. But, the value of annotation file fields specified in the annotation expression are not allowed with duplicated value. Furthermore, we provide the user interface tool (named gvaf_creation_annotation_file) to generate an annotation file (see page 30).

- Annotation expression

  ```
  Input file field==annotation file field
  ```

  Example:

  - Field names of Input file: “Chr, Start, End, Ref, Alt, Gene.refGene”
  - Field names of an annotation file: “OMIM.Gene, disease name, alternative titles”

  ```
  -annotate Gene.refGene==OMIM.Gene
  ```

  Using this annotation expression, you can annotate each variant form the input files if the value of Gene.refGene field is equal to the value of OMIM.Gene field of a certain line from an annotation file, by attaching all fields from an annotation file at the end of the line of the corresponding variant from the input files.

This category includes the options related to the filtering function. You can extract the interesting/relevance variants from the input files using the filtering function. GVAF provides a flexible filtering function by applying the filtering rule containing the expressions consisting of the various operators (see page 8-9) in all fields of the input file as a filtering rule without special limitation. In addition, GVAF can process multiple filtering rules at same time and provides the ability to apply the different input files to each filtering rule using the concept of group.

To execute the filtering module, GVAF requires two types of file: FilteringRule file (called FR file) and FieldDefinition file (called FD file).

- FR file

  This is designed to define the execution method for the filtering module and consists of three components.

  - BuildingPrimarykey: This is designed for the expressions with a set operator and set to a list of input file field names as value for identifying each variant in the filtered result files. The expressions with a set operator compare the variants between the filtered result files by referring the value of BuildingPrimarykey. This will be automatically set to ““Chr, Start, End, Ref, Alt” as the default value if this is not set.
  - InputGroupAlias: This is used to define the groups designed to apply the different input file per each filtering rule. GVAF uses the input files by “-input” option set in gvaf tool to define the groups, and each group includes one or more files or directories.

    - How to create a group

    ```
    Directory path or file path, Directory path or file path, … = group name
    ```

    Each line specifies a group, where a group consists of multiple files or directories. No duplicate is allowed for group names. Comma (,) is used as a delimiter between the paths of the files or directories.

    ```
    <InputGroupAlias
    example_dir1==group1
    example_dir1, example_dir2==group2
    example_input1.txt==group3
    example_input1.txt, example_input2.txt==group4
    >
    ```

    Figure 3. An example of InputGroupAlias.

    Figure 3 shows an example of InputGroupAlias consisting of four groups (e.g. group1~4) to demonstrate the definition method for the multiple groups. Group1 and group2 are defined by inputting the directory and these groups are set to all files in defined directories (e.g., example_dir1 or example_dir2). Group 3 and group 4 are defined by entering the file, and the defined files (e.g., example_input1.txt and example_input2.txt) are organized to each group.

  - FilteringRule: This is used to define one or more filtering rules, and you can create the filtering rules using the expressions consisting of three operator types such as comparison operators, logical operators, and set operators.

    - Comparison operators

This is used to define a filtering rule at a field of the input or annotated files using a comparison operator. GVAF currently supports three field data types (e.g., S, I and D).

Operators	Field data type	Description
e	S (String), I (integer), D (double)	Equal to
ne	S (String), I (integer), D (double)	Not equal to
c	S (String)	Contains
nc	S (String)	Does not contain
lt	I (integer), D (double)	Less than
lte	I (integer), D (double)	Less than or equal to
gt	I (integer), D (double)	Great than
gte	I (integer), D (double)	Great than or equal to

Table 3. The available comparison operations.

- Logical operators

  This is used to combine the expressions for defining a filtering rule using the logical operators using symbols such as "* and &" (AND) and "| and +" (OR). We provide the two types of logical operators: The inner field logical operator and the outer field logical operator.

  ✓ The inner field logical operator

  This is used to combine the expressions applied for one field using symbols: "&" (AND) and "|" (OR).

  ✓ The outer field logical operator

  This is used to combine the expressions applied for multiple fields using the symbols: "*" (AND) and "+" (OR).

  The logical operators can be defined according to the type of combined expression. We will explain detailed instructions to define the expressions with the logical operator using some example (see page 9-10).

- Set operators

  This is used to define the expression of a filtering rule for performing the additional filtering function between the filtered result files from the multiple filtering rules including the expressions consisting of comparison operators and logical operators. The expressions including a set operator can be defined using two or more filtering rules with one or more result files. The set operators can be defined using predefined strings such as union, difference and intersection without case sensitive and the filtering rules with the filtered result files are defined using the alias. We will explain detailed instruction to define the expression with the set operator using some example (see page 10-13).

The filtering rules can be defined as below.

```
Expression==alias;group1,group2,...
```

Each line specifies a filtering rule, where a filtering rule consists of an expression, the

group information, and alias. No duplicate is allowed for a filtering rule alias. The group information applied a filtering rule is specified using ";" (semicolon) and can define the multiple groups using "," (comma). The filtering rules with no group information are automatically set to a default group that contains all files established by "-input" option of gvaf tool. The expressions are composed of the filters that can be apply to the input files and must be defined according to the following instruction.

- How to define the expressions

  ✓ The expressions consisting of comparison and logical operators

  This is used to define a filtering rule for the fields of the input files or annotated result files and can be defined using comparison and logical operators without any limitation. Using the comparison and logical operators, you can define the expression that compares one or more values in one field or one or more value in several fields. You can create the expressions as below.

  ```
  Field name: compared value%comparison operator [inner field logical operator] ...
  ```

  Each expression for one field consists of three main components.

  1. Field name: This is used to specify a filtered field name of the input or annotated result files.
  2. Compared value: This is used to specify the compared value for applying a comparison operator
  3. Comparison operator: This is used to specify the comparison operator for applying filtered field.

  Furthermore, you can generate an expression containing two or more comparison operators for one field using the inner field logical operators such as "& (AND) or | (OR)". We prepared two examples.

  ```
  (Chr:chr1%e)
  ```

  This example shows an expression containing one comparison operator. Using this expression, you can extract the lines including the value of "chr1" in the field named "Chr" from the input or annotated result file.

  ```
  Func.refGene:UTR3%e|splicing%e
  ```

  This example shows an expression containing multiple comparison operators using the inner field logical operator such as "& (AND) and | (OR)". Using the expression, you can extract the lines including the values of "UTR3" or "splicing" in the field named "Func.refGene" from the input or annotated result file.

  ```
  (Chr:chr1%e)*(Func.refGene:UTR3%e|splicing%e)
  ```

  You can also generate an expression combined with two expressions of above examples using the outer field logical operators such as "* (AND) and + (OR)". Each expression of above examples can be identified using "()". Using this example expression, you can extract the line including the value of "chr1" in the field named "Chr" and the values of "UTR3" or "splicing" in the

field named “Func.refGene” from the input or annotated result file.

- ✓ The expressions consisting of a set operator

  This is used to perform the filtering function between the filtered result files of two or more individuals by applying the set operators. You can define the expression using the set operator and the alias of the filtering rules as below.

  ```
  #set operator:filtering rule alias1, filtering rule alias2, …
  ```

  Each expression with a set operator consists of two main components.

  1. Set operator: This is used to specify the type of set operators with “#”. The set operator can be defined using the strings such as union, difference and intersection without case sensitive.
  2. Filtering rule alias: This is used to specify the two or more filtering rules for applying the filtering function and can be defined using “,” (comma) as delimiter. Each filtering rule is composed of comparison and logical operators with the result files.

  This expression produces the final result files by applying a set operator based on the first defined filtering rule. You should define the proper filtering rules in this expression depending on a first defined filtering rule. A first defined filtering rule with one filtered result file can be applied using the remaining filtering rules with one filtered result file. Otherwise, a first defined filtering rule can be applied using the filtering rule regardless of number of the result files. But, the remaining filtering rules with multiple filtered results must be identical in the same original input files of a first defined filtering rule.

  We prepared some examples to show the applicable scenarios.

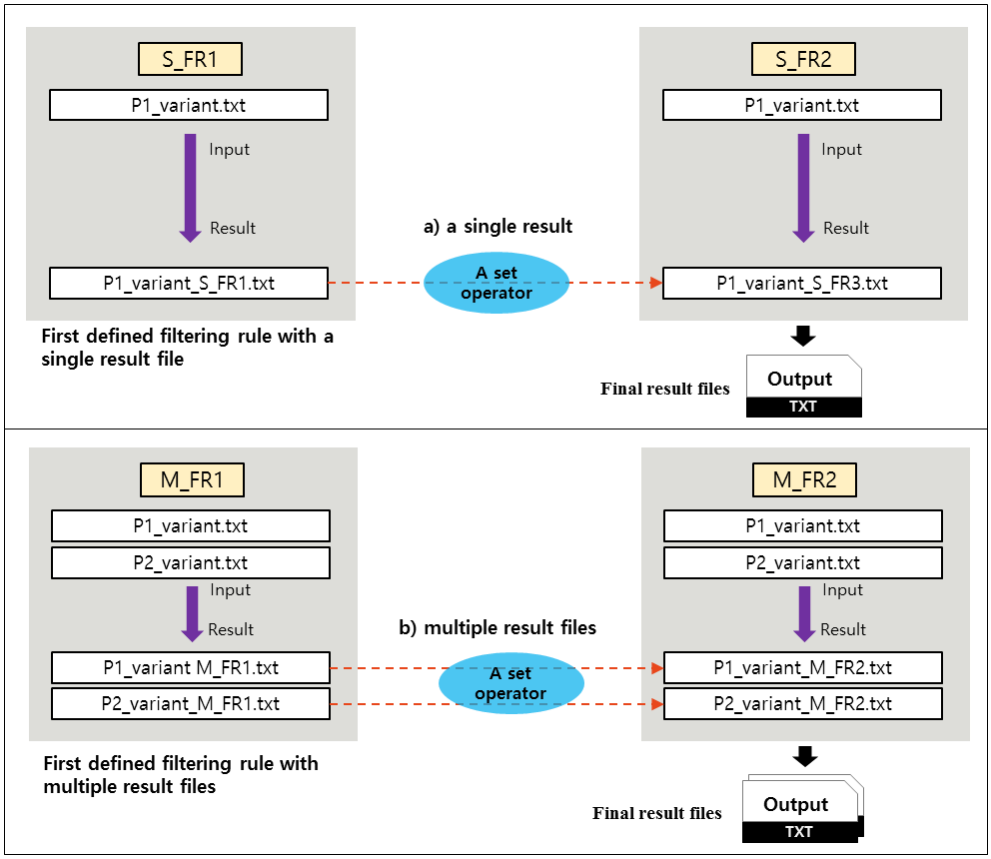


Figure 4. An example for applying a set operator between filtered result files from the same samples.

Figure 4 shows an example of applying a set operator between the filtered result files of same individuals. Figure 4 (a) shows an example of how to generate a final result file when two filtering rules (i.e., S_FR1, S_FR2) have the result files from the same input file (i.e., P1_variant.txt). Figure 4 (b) shows an example of how to generate the final result files when two filtering rules (i.e., M_FR1, M_FR2) have the result files from two identical input files (i.e., P1_variant.txt, P2_variant.txt). At this time, you must define the expression using the filtering rules including completely identical input files.

Moreover, you can execute the filtering function using a set operator between the filtered result files of the filtering rules using a filtered result file of a specific person. Using this filtering function, you can fine the causative variants of patient related to disease by comparing the genetic variants of parents or siblings.

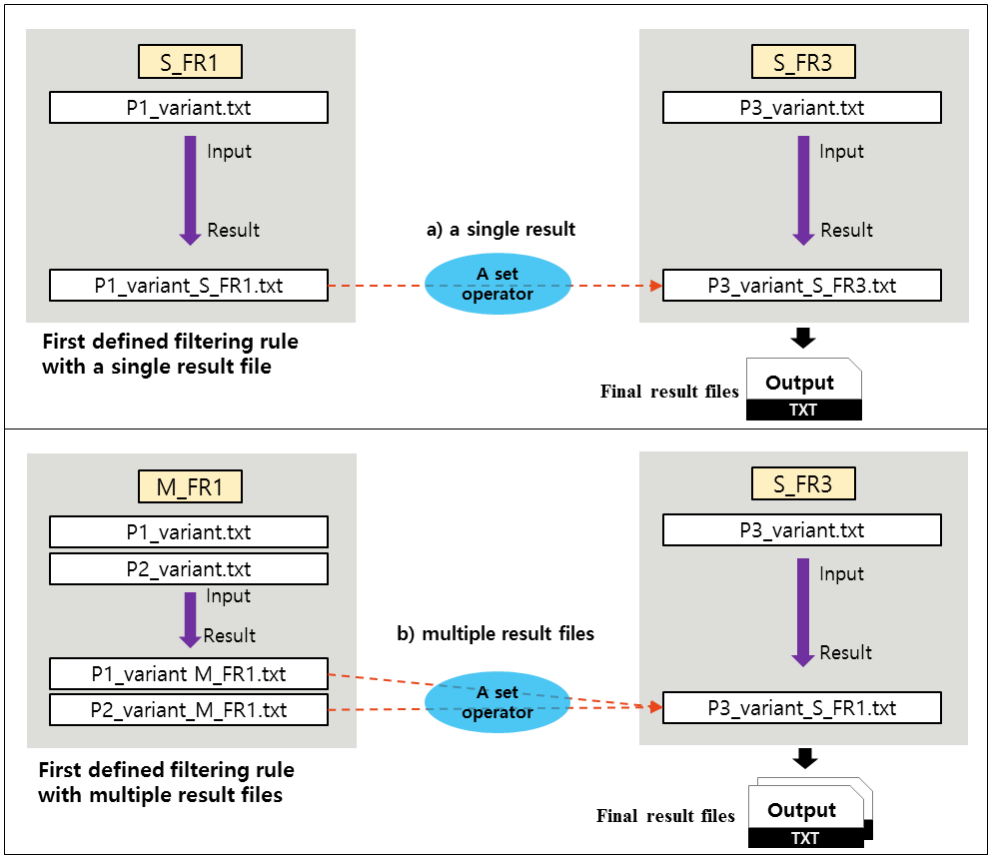


Figure 5. An example for applying a set operator between the filtered result files from the other samples.

Figure 5 shows an example of applying a set operation between the filtered result file of different individuals. Figure 5 (a) shows an example of how to generate a final result file when two filtering rules (i.e., S_FR1, S_FR3) include one result file from the input files (i.e., P1_variant.txt, P1_variant.txt). In Figure 5 (b), we show example of how to generate the final result files when two filtering rules (i.e., M_FR1, S_FR3) is not equal to the number of result files. The final result files are generated by executing the filtering function with a set operator based on two result files (i.e., P1_variant M_FR1.txt, P2_variant M_FR1.txt) in M_FR1 using one result file (i.e., P3_variant_S_FR1.txt) in S_FR3. At this time, you must define the expression using the filtering rules with one result files when a first defined filtering rule includes multiple input files.

We prepared an example to perform the filtering function with a set operator by applying three or more filtering rules for advanced usage.

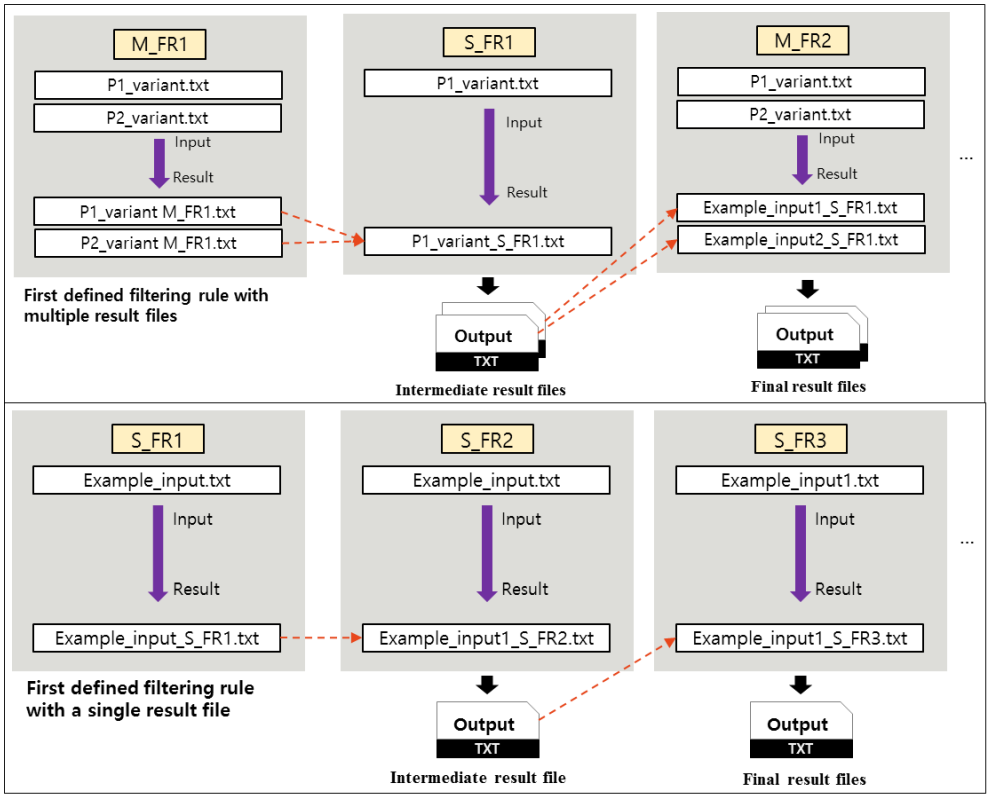


Figure 6. An example for applying a set operator between the result files from three or more filtering rules.

You can define the expression with a set operator and several filtering rules depending on the result file number of a first defined filtering rule such as an example in Figure 6. All intermediate result files are only used to perform the filtering function and not stored to local file system. You can find the final result files on local file system.

Furthermore, GVAF provides the multiple-valued operators for handling the fields containing the multiple values.

- Multiple-valued operators (MOP)
  - ✓ all: If you set “all”, the expression will be true if all field satisfy the expression.
  - ✓ any: If you use “any”, the expression will be true if any field stratifies the expression.

```
(Func.refGene, all:UTR3%e|splicing%e)
(Func.refGene, any:UTR3%e|splicing%e)
```

Figure 7. The expression examples using MOP.

MOP can be applied to a single field containing the multiple values in the expression consisting of comparison and logical operators. Figure 7 shows the expression example using MPOs such as “all” and “any”. Two expressions are the same except for MOP in figure 7. But, the meanings of two expressions are completely different. We assumed that field named “Func.refGene” has multiple values. Using filtering rule defined the first expression example applying “all”, you

can extract the input file lines that all values of “the field named “Func.refGene” are UTR3” or “splicing”. Using filtering rule defined the first expression example applying “any”, you can extract the input file lines that any values of “the field named “Func.refGene” are UTR3” or “splicing”. The fields containing the multiple values automatically set “any” as the default operator.

We demonstrate how to create a FR file through an example.

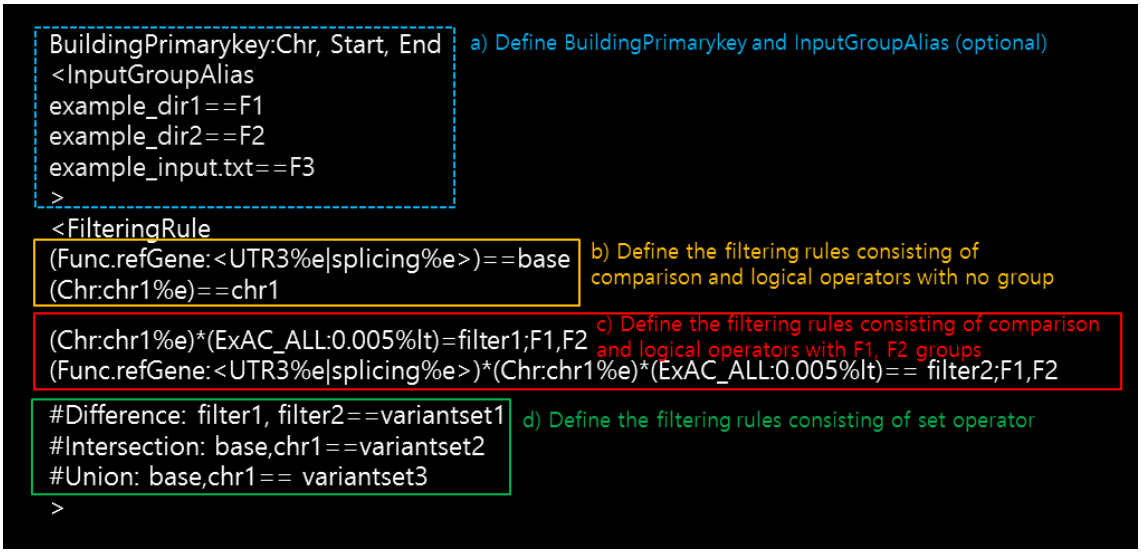


Figure 8. the example of a FR file.

BuildingPrimarykey, InputGroupAlias, FilteringRule and includes 7 filtering rules. Figure 8 (a) can be skipped selectively if you want to apply default value. Otherwise, you must define BuildingPrimarykey and InputGroupAlias as shown in figure 8 (a). Figure 8 (b) indicates two filtering rules including the expression consisting of comparison and logical operators with no group. Figure 8 (c) indicates the filtering rules two filtering rules including the expression consisting of comparison and logical operators with two groups (i.e., F1, F2). Figure 8 (d) shows three filtering rules including the expression specified in a set operator. The result files are generated depending on the group information of each filtering rule in a FR file.

- FieldDefinition file (FD file)

  A FD file is used to input the information of the field defined in the filtering rules of a FR file and is composed of three fields.

  - Name: This is used to enter the input file field name specified in a FR.
  - Type: This is used to define the field data type using a character (e.g., S, I, D).
  - Delimter(req): This is used to define the delimiter of the field including multiple values using the regular expression and can be skipped for the field having a single value.

Name	Type	delimiter(req)
Chr	S	
Func.refGene	S	;
Gene.refGene	S	,
ExAC_ALL	D	

Figure 9. An example of a FD file.

The figure 9 shows an example of a FD file to define the field information including the fields defined in the filtering rules in a FR file shown in the figure 8.

GVAF provides additional function (called ListFiltering) that is applied to all filtering rule in a FR file to identify the line containing the variants associated with a certain disease group or disease or a gene set associated with a specific disease.

- ListFiltering

  ListFiltering performs additional function on the filtered result files using a file (called a ListFiltering file). A ListFiltering file includes the values associated with a certain disease group or a gene set associated with a specific disease. Using the ListFiltering-related options and a ListFiltering file, you can distinguish whether or not each line in the filtered result files has the variant belonging to the value in a ListFiltering file. As a result, you can get two types of result files such as a list file or a unlist file per the filter result files.

- A list file: This extracts the lines from a filtered result file with the value belonging to a ListFiltering file.
- A unlist file: This extracts the lines from a filtered result file without the value belonging to a ListFiltering file.

Figure 10. An example of a ListFiltering file.

The figure 10 shows the example of a ListFiltering file to define the values. Each value be defined on one line. You can define the values as you want easily for executing ListFiltering function. In figure10, four values are defined in a ListFiltering. Using a ListFiltering file, you can get two result files such as list or unlist per the filtering rules in a FR file. A list file has the variants including ACTG1, AIPL1, ALS5, ARWH1 and a unlist file without the variants including ACTG1, AIPL1, ALS5, ARWH1.

Format

This category includes the options related to the format function that is used to change the output file format. Using this option in this category, you can print the specific fields or change the order of the printing fields using a format file.

- Format file

  A format file is used to define the output file format using the field name of an input file. You can generate a format file without the specific field or components using the field name easily. A format file can be defined by inserting the field names in the input files or annotated result file in the order of the printed fields. Also, you can change the printed field names using the simple definition format as defined "input field name ==>modified field name".

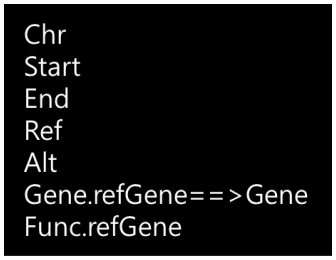


Figure 11. An example of format file.

Each line specifies the printed fields of the input or annotated files. The figure 11 shows an example of a format file to define customized output file format. Using this format file, you can get the result files consisting of “Chr,Start,End,Ref,Alt,Gene,Fun.refGene”.

Execution examples with gvaf

We provide some examples and dummy data to illustrate how to use gvaf for exploring the genetic variants. We assume that the commands for applying all examples are performed using the files and directories belonging to dummy data stored in specific directory named “dummydata” as input on the home directory of GVAF. The dummy data with Annovar-like output file format is used as input in all examples and consists of two directories (i.e., example_dir1, and example_dir2) and seven files (i.e., example_input.txt, example_dir1_1.txt, example_dir1_2.txt, example_dir1_3.txt, example_dir2_1.txt, example_dir2_2.txt, and example_dir2_3.txt). You can download gvaf program and dummy data together.

Usage

gvaf -input FILE/DIRECTORY [-outputdir DIRECTORY] [-threads INTEGER] [PREPROCESIING OPTIONS] [ANNOTATION OPTIONS] [FILTERING OPTIONS] [FORMAT OPTIONS]

An execution example using the preprocessing category

GVAF can add the genotype-related fields to input files by converting the VCF derived genotype data using the preprocessing module. The preprocessing module can be executed by setting the options in the preprocessing category. We describe the usage of the options in the preprocessing category using an example.

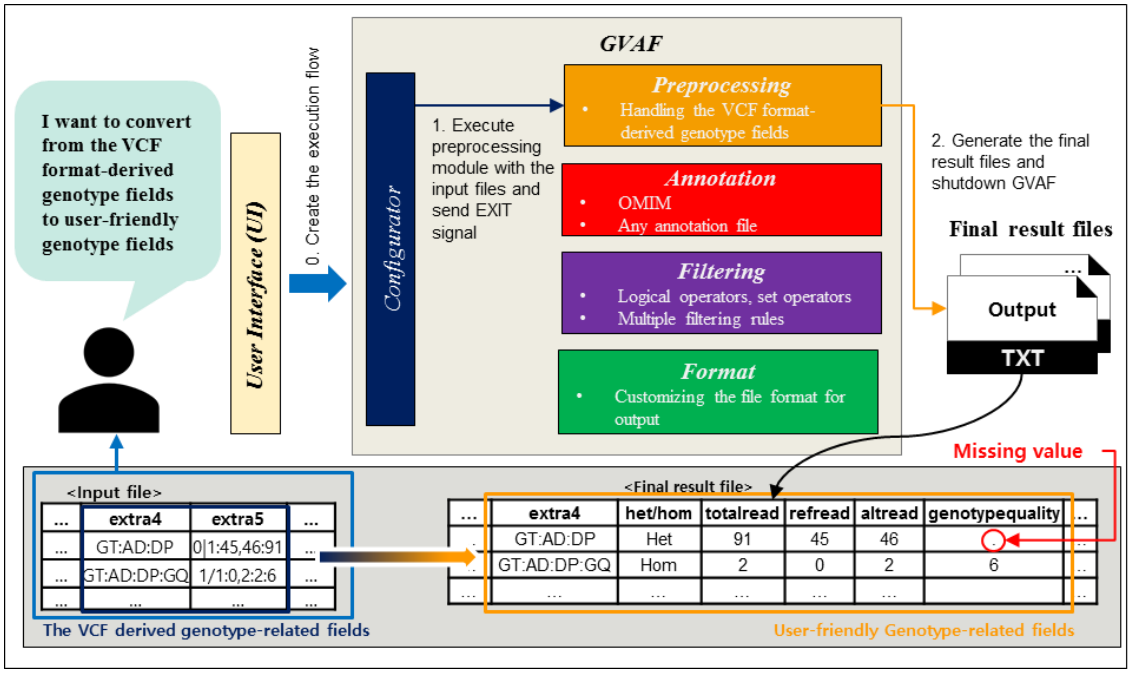


Figure 12. An execution example using preprocessing module.

Figure 12 shows the process of transforming the VCF derived genotype data to the user-friendly genotype-related fields such as het/hom, totalread, refread, altread, genotypequality. This example is performed using the files in dummydata as input and the options belonging to preprocessing category by setting the field names including the VCF derived genotype-related data and VCF derived genotype-related data format. The files in dummydata with ANNOVAR-like output file includes two fields with VCF derived genotype-related data and VCF derived genotype-related data format. Initially, these field names including the VCF derived genotype-related data are the null values in input file. GVAF renames the name of fields with the null values in sequence using the letter (extra1~n) before using as input to each module. Thus, the file name including VCF derived genotype-related data format is renamed as extra4 and the field name including VCF derived genotype-related data is renamed as extra5.

You can execute an example in figure 12 using the following command.

- Using one file as input

```
gvaf -input dummydata/example_input.txt -genotype_field "extra5" -genotype_format_field "extra4“
-outputdir result
```

Using this command, you can set one file (named example_input.txt in dummydata directory) as input using “-input” option. Two options in preprocessing category such as "-genotype_field" and "-genotype_format_field" are set. The "-genotype_field" option is set by entering the field name (i.e., extra5) including genotype-related data in input file. The "-genotype_format_field" option is set by entering the field name (i.e., extra5) including genotype-related data format in input file. The “-outputdir” option is set to the directory path of result, and the final result file is stored in preprocessing directory of output location as below.

- Final result file (from one input file named example_input.txt)

  preprocessing_example_input.txt

- Using multiple files as input

```
gvaf -input dummydata/example_input.txt -genotype_field "extra5" -genotype_format_field "extra4“
-outputdir result
```

We will only mention the description of “-input” option and the final result files. You can find the description of the remaining options in above command. Using this command, the multiple files belonging to example_dir1, example_dir2 in dummydata directory are used as input using “-input” option. The example_dir1 and example_dir2 directories have six files as below.

```
./example_dir1:
example_dir1_1.txt  example_dir1_2.txt  example_dir1_3.txt

./example_dir2:
example_dir2_1.txt  example_dir2_2.txt  example_dir2_3.txt
```

The final result files are stored in preprocessing directory of output location as below.

- Final result file (from six input files in two directories)

  preprocessing_example_dir1_1.txt, preprocessing_example_dir1_2.txt
  preprocessing_example_dir1_3.txt, preprocessing_example_dir2_1.txt
  preprocessing_example_dir2_2.txt, preprocessing_example_dir2_3.txt

Execution example using annotation category

GVAF can add the additional fields including information about each variant in the input files using an annotation file based on annotation expression. The annotation module can be executed by setting the options in the annotation category and an annotation file and annotation expression. You can get the detail of an annotation file and annotation expression (see page 7). More specifically, each line in input file and the lines of an annotation file with the value satisfied in annotation expression are combined into a single line, and combined lines are stored to a result file. If the value of input file field specified in annotation expression does not match any values of annotation file field in annotation expression, the annotation module creates the line consisting of the line in input file and meaningless line including the missing value (“—“), and created lines are stored to a result file. We describe the usage of the options in the annotation category via two examples.

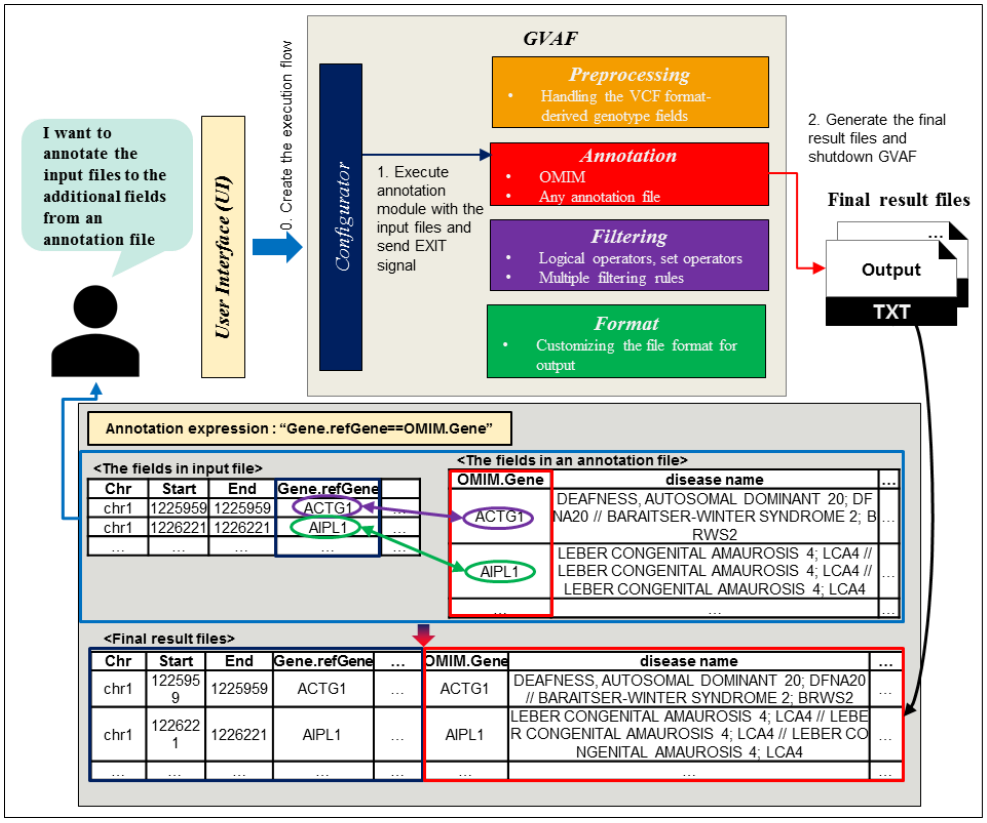


Figure 13. The execution example of annotation module using OMIM gene annotation file.

Figure 13 shows the process of annotating additional fields to the input file based on annotation expression using OMIM gene annotation file. OMIM gene annotation file is composed of 10 fields such as "OMIM.Gene, disease name, etc." and is provided by default without a specific annotation file. Using this annotation expression, the annotation module combines the line of input file and annotation file when the value of Gene.refGene field in input file is equal to the value of OMIM.gene field in annotation file, and combined lines is stored to the final result file. In other cases, the annotation module creates the line consisting of the line in input file and meaningless line including the missing value ("—"), and created lines are stored in the final result file.

You can execute an example in figure 13 using the following command.

- Using one file as input

```
gvaf –input dummydata/example_input.txt -annotate "Gene.refGene==OMIM.Gene" -annotation_file "omim" -outputdir result
```

Using this command, you can set one file (named example_input.txt in dummydata directory) as input using "-input" option. Two options in annotation category such as "-annotate" and "-annotation_file" are set. The "-annotate" option is set by entering "Gene.refGene==OMIM.Gene" as the annotation expression. The "-annotation_file" option is set by entering "omim" as an annotation file because GVAF provides the OMIM gene annotation by default. Using "-outputdir" option, you set result directory as output location for storing the result files. The final result file is stored in annotation directory of output location as below.

- Final result file (from one input file named example_input.txt)

  Annotated_example_input.txt

- Using the multiple files as input

```
gvaf -input "dummydata/example_dir1, dummydata/example_dir2" -annotate "Gene.refGene==OMIM.Gene" -annotation_file "omim" -outputdir result
```

We will only mention the description of "-input" option and the final result files. You can find the description of the remaining options in above command. Using this command, the multiple files belonging to example_dir1, example_dir2 in dummydata directory are used as input using "-input" option. The example_dir1 and example_dir2 directories have six files as below.

```
./example_dir1:
example_dir1_1.txt  example_dir1_2.txt  example_dir1_3.txt

./example_dir2:
example_dir2_1.txt  example_dir2_2.txt  example_dir2_3.txt
```

All annotated result files are stored in annotation directory of output location as below.

- Final result file (from six input files in two directories)

  Annotated_example_dir1_1.txt, Annotated_example_dir1_2.txt
  Annotated_example_dir1_3.txt, Annotated_example_dir2_1.txt
  preprocessing_example_dir2_2.txt, Annotated_example_dir2_3.txt

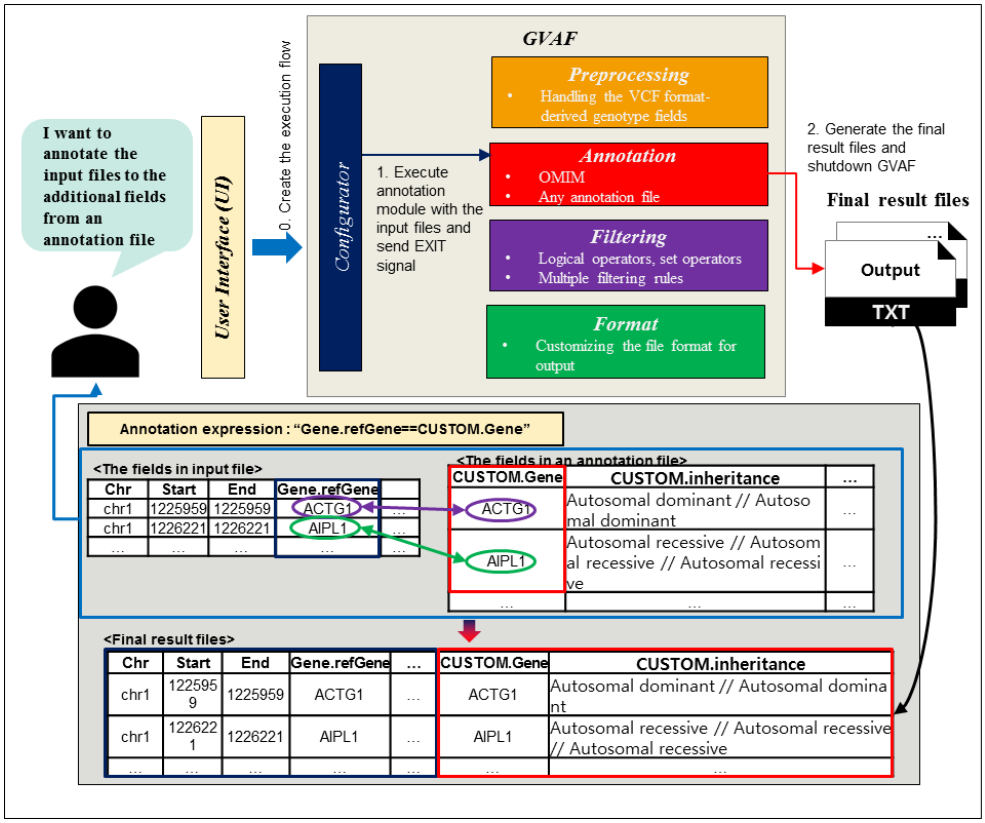


Figure 14. An execution example of annotation module using custom gene annotation file.

Figure 14 shows the process of annotating additional fields from custom gene annotation file to the input file based on annotation expression. We created an annotation file (named custom_annotation.txt) to execute the annotation module using a file consisting of the arbitrary fields as annotation file. This annotation file is stored in refdata directory of the dummydata directory.

The custom gene annotation file is composed of two fields such as “CUSTOM.Gene, CUSTOM.inheritance”. Using this annotation expression, the annotation module combines the line of input file and annotation file when the value of Gene.refGene field in input file is equal to the value of CUSTOM.gene field in annotation file, and combined lines is stored to the final result file. In other cases, the annotation module creates the line consisting of the line in input file and meaningless line including the missing value (“—“), and created lines are stored in the final result file.

You can execute an example in figure14 using the following command.

- Using one file as input

```
gvaf -input dummydata/example_input.txt -annotate "Gene.refGene==CUSTOM.Gene" -annotation_file
custom_annotation.txt -outputdir result
```

Using this command, you can set one file (named example_input.txt in dummydata directory) as input using “-input” option Two options in annotation category such as “-annotate” and “-annotation_file” are set. The “-annotate” option is set by entering "Gene.refGene==CUSTOM.Gene" as the annotation expression. The “-annotation_file” option is set by entering the file path of custom_annotation.txt as an annotation file. Using “-outputdir” option, you set result directory as output location for storing the result files. The final result file is stored in annotation directory of output location as below.

- Final result file (from one input file named example_input.txt)

  Annotated_example_input.txt

- Using multiple files as input

```
gvaf -input "dummydata/example_dir1, dummydata/example_dir2" -annotate
"Gene.refGene==CUSTOM.Gene" -annotation_file custom_annotation.txt -outputdir result
```

We will only mention the description of “-input” option and the final result files. You can find the description of the remaining options in above command. Using this command, the multiple files belonging to example_dir1, example_dir2 in dummydata directory are used as input using “-input” option. The example_dir1 and example_dir2 directories have six files as below.

```
./example_dir1:
example_dir1_1.txt  example_dir1_2.txt  example_dir1_3.txt

./example_dir2:
example_dir2_1.txt  example_dir2_2.txt  example_dir2_3.txt
```

All annotated result files are stored in annotation directory of output location as below.

- Final result file (from six input files in two directories)

  Annotated_example_dir1_1.txt, Annotated_example_dir1_2.txt
  Annotated_example_dir1_3.txt, Annotated_example_dir2_1.txt
  Annotated_example_dir2_2.txt, Annotated_example_dir2_3.txt

An execution example using filtering category

GVAF extracts the lines with the variants including the matched expressions of the filtering rules from the input or annotated files using the filtering module. The filtering module is performed by

setting the options in the filtering category and two types of files such as FilteringRule file (called a FR file) and FieldDefinition file (called a FD file). You can find the detail of FR file (see page 7) and FD file (see page 14). The filtering module filters each line in the input files by referring the filtering rules in a FR file and the field information in a FD file and provides additional function called ListFiltering function. ListFiltering can be optionally executed and is used to identify whether each line in the filtered result files has the variants associated with a certain disease group or gene set. You can get the information about the ListFiltering function (see page 15). We describe the usage of the options in filtering category using two examples as below.

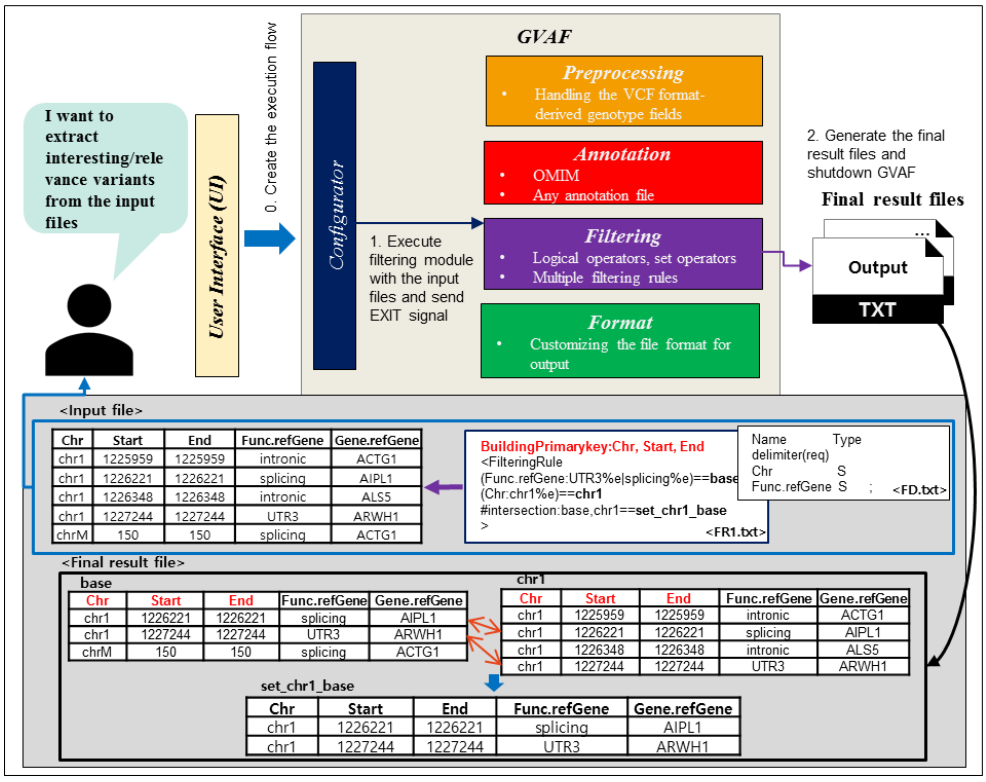


Figure 15. The execution example for filtering module without ListFiltering step.

The figure 15 shows the process of filtering each line of the input files containing the variant information based on the filtering rule defined in a FR file. We created two types of files such as a FR file (named FR.txt) and a FD (named FD.txt) for executing the filtering module. A FR file contains three filtering rules with aliases such as base, chr1, and set_chr1_base and a FD file contains two field information specified in the expression of the filtering rules. These files are stored in refdata directory of the dummydata directory. Two filtering rules (called base and chr1) create the filtered result files based on the input files by applying the expressions consisting of comparison and logical operators. A filtering rule (called set_chr1_base) creates the filtered result files by applying the expression consisting of a set operator between the filtered result files of base and chr1. You can get three filtered result files per an input file.

You can execute an example in figure 15 using the following command.

- Using one file as input

```
gvaf -input dummydata/example_input.txt -FilteringRule_File FR.txt –FieldDefinition_File FD.txt -outputdir result
```

Using this command, one file (named example_input.txt in dummydata directory) is used as

input using “-input” option. Two options in filtering category such as “-FilteringRule_File” and “–FieldDefinition_File” are set. The “-FilteringRule_File” option is set by entering the file path of FR.txt as a FR file. The FieldDefinition_File” option is set by entering he file path of FD.txt as a FD file. The result directory is set as output location for storing the result files. All filtered result files are stored in filtering directory of output location as below.

- Final result file (from one input file named example_input.txt)

  Filtered_example_input_base.txt, Filtered_example_input_chr1.txt
  Filtered_example_input_set_chr1_base.txt

You can get three filtered result files because the filtered result files are generated by applying three filtering rules in a FR file based on an input file.

- Using multiple files as input

```
gvaf -input "dummydata/example_dir1, dummydata/example_dir2“ -FilteringRule_File FR.txt –
FieldDefinition_File FD.txt -outputdir result
```

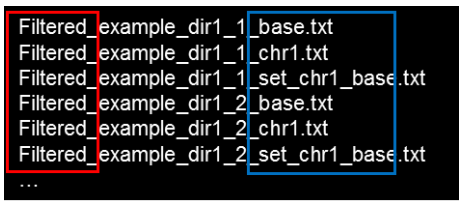
We will only explain “-input” option and the final result files. You can get the description of the remaining options in above command. Using this command, multiple files belonging to example_dir1, example_dir2 in dummydata directory are used as input using “-input” option. The example_dir1 and example_dir2 directories have six files as below.

```
./example_dir1:
example_dir1_1.txt  example_dir1_2.txt  example_dir1_3.txt

./example_dir2:
example_dir2_1.txt  example_dir2_2.txt  example_dir2_3.txt
```

All filtered result files are stored in filtering directory of output location as below.

- Final result file (from six input files in two directories)



You can get 18 filtered result files because the filtered result files are generated by applying three filtering rules in a FR file based on 6 input files.

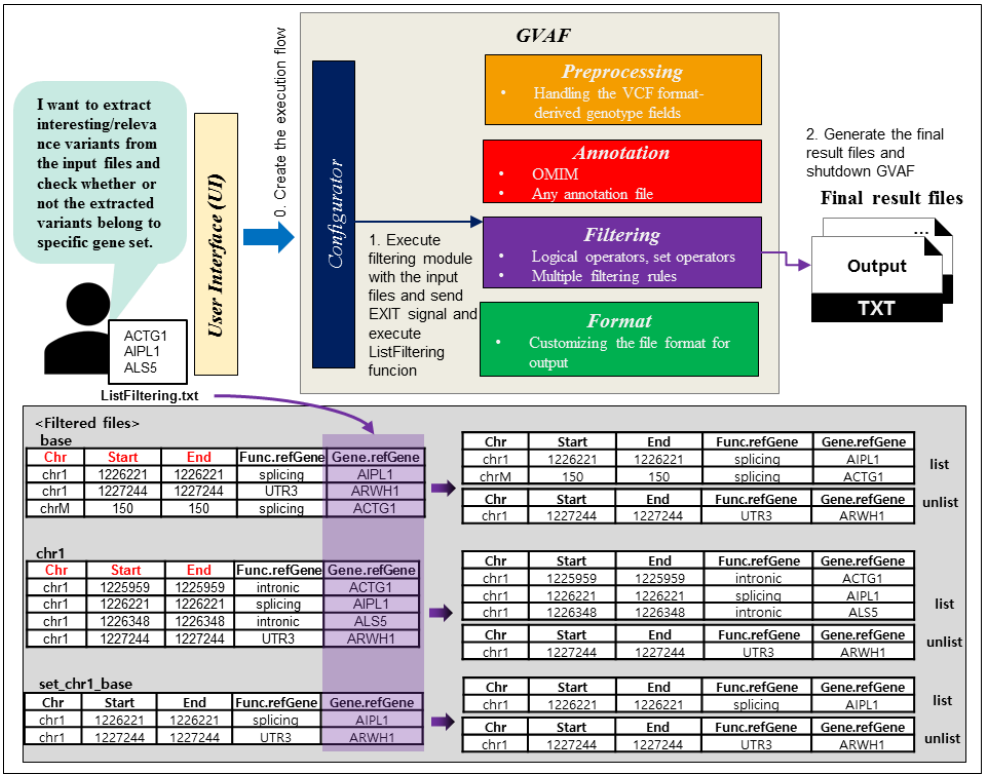


Figure 16. The execution example for filtering module with ListFiltering step.

Figure 16 shows the process of performing the ListFiltering function for the filtered result files. The ListFiltering function identifies whether each line in the filtered result files has the value belonging to the values in a ListFiltering file by referring to the value of a certain field. We created a ListFiltering file including 3 values for executing this example. A ListFiltering file (named ListFiltering.txt) is stored in refdata directory of dummaydata directory. ListFiltering function is performed using the filtered result files shown in an example in figure 15 and ListFiltering.txt as a ListFiltering file. You can get two types of result files such as a list or unlist files per a filtered result file. Each line of a filtered result file with the value belonging to the value in a ListFiltering file is stored to as list file. Each line of a filtered result file without the value belonging to the value in a ListFiltering file is stored as a unlist file.

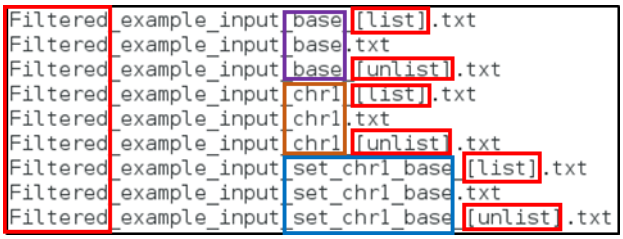
You can execute an example in figure 16 using the following command.

- Using one file as input

```
gvaf -input dummydata/example_input.txt -FilteringRule_File FR.txt –FieldDefinition_File FD.txt -outputdir result -ListFiltering_field "Gene.refGene" -ListFiltering_file ListFiltering.txt
```

Using this command, one file (named example_input.txt in dummydata directory) is used as input using “-input” option. “-FilteringRule_File” and “FieldDefinition_File” options set the same value as an example in figure 15 (see page 23). In addition, two options associated with ListFiltering function such as "-ListFiltering_field" and "-ListFiltering_file" options are set. The "-ListFiltering_field" option is set by entering a field name that is referenced when performing ListFiltering function in the filtered result files. The "-ListFiltering_file" option is set by entering a file path of ListFiltering.txt. The result directory is set as output location for storing the result files. All filtered result files are stored in filtering directory of output location as below.

- Final result file (from one input file named example_input.txt)



- Using multiple files as input

```
gvaf -input "dummydata/example_dir1, dummydata/example_dir2“ -FilteringRule_File FR.txt –FieldDefinition_File
FD.txt -outputdir result -ListFiltering_field "Gene.refGene" -ListFiltering_file ListFiltering.txt
```

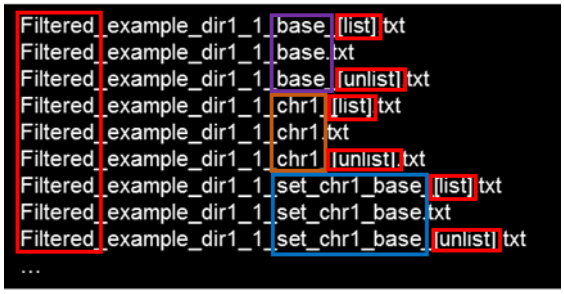
We will only explain “-input” option and the final result files. You can get the description of the remaining options in above command. Using this command, the multiple files belonging to example_dir1, example_dir2 in dummydata directory are used as input using “-input” option. The example_dir1 and example_dir2 directories have six files as below.

```
./example_dir1:
example_dir1_1.txt  example_dir1_2.txt  example_dir1_3.txt

./example_dir2:
example_dir2_1.txt  example_dir2_2.txt  example_dir2_3.txt
```

All filtered result files are stored in filtering directory of output location as below.

- Final result file (from six input files in two directories)



An execution example using format category

GVAF can export output file with customized file format using the format module. The format module is performed by setting the options in the format category and a format file. You can get a detail of a Format file (see page 15). Using the format module, you can apply the various situations. For example, you can extract the specific fields in annotated result file or input files and change the order of the printing fields and modify the field name of the printing fields.

We explain the usage of the options in format category using an example as below.

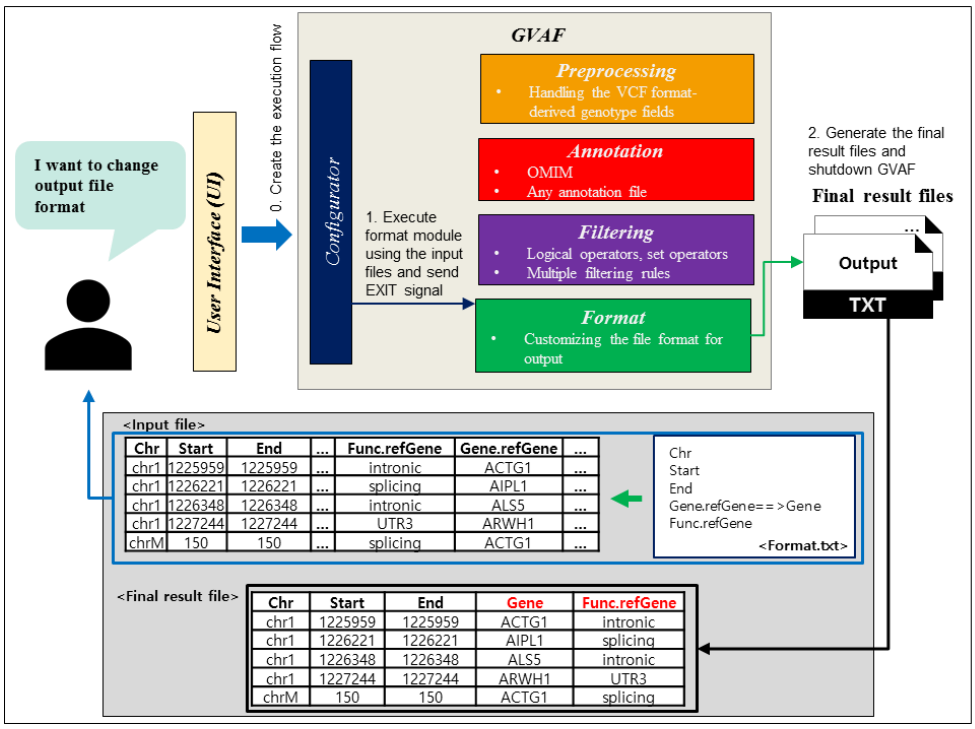


Figure 17. The example of customized output file creation using a format file.

Figure 17 shows the process of generating the result files with customized file format using the format module. The customized file format is defined using a format file. You can find a detail of a format file (see page 15). We created a format file (named Format.txt) containing output format of the 5 fields in the input files for executing this example. This file is stored in refdata directory of the dummydata directory. The input files consist of 57 fields such as "Chr, Start, End, Func.refGene, Gene.refGene, etc.". The format module extracts the 5 fields (i.e., Chr, Start, End, Func.refGene, Gene.refGene defined in a format file) form the input files and prints extracted fields in order defined in a format file. As a result, the field named “Gene.refGene” is changed from “Gene.refGene” to “Gene”. You can get the final result files consisting of Chr,Start,End,Ref,Alt,Gene,Fun.refGene”.

You can execute an example in figure 17 using the following command.

- Using one file as input

```
gvaf -input dummydata/example_input.txt -format_file Format.txt -outputdir result
```

Using this command, one file (named example_input.txt in dummydata directory) is used as input using “-input” option. The “-format_file” option in the format category is set by entering the file path (Format.txt) as a format file. The result directory is set as output location for storing the result files. A final result file is stored in format directory of output location as below.

- Final result file (from one input file named example_input.txt)

  Formatted_example_input.txt

- Using multiple files as input

```
gvaf -input “dummydata/example_dir1, dummydata/example_dir2” -format_file Format.txt -outputdir result
```

We will only mention the description of “-input” option and the final result files. You can find the description of the remaining options in above command. Using this command, the multiple files belonging to example_dir1, example_dir2 in dummydata directory are used as input using “-input” option. The example_dir1 and example_dir2 directories have six files as below.

```
./example_dir1:
example_dir1_1.txt  example_dir1_2.txt  example_dir1_3.txt

./example_dir2:
example_dir2_1.txt  example_dir2_2.txt  example_dir2_3.txt
```

The final result files are stored in format directory of output location as below.

- Final result file (from six input files in two directories)

  Formatted_example_dir1_1.txt, Formatted_example_dir2_1.txt,
  Formatted_example_dir1_2.txt, Formatted_example_dir2_2.txt
  Formatted_example_dir1_3.txt, Formatted_example_dir2_3.txt

An execution example using several categories

GVAF can execute the multiple modules at once using the options in several categories. Each module is executed in order according to the predefined priority. At this time, all modules except the initially executed module use the result files of the previously executed module as the input files. The result files of each module are stored in individual directory of output directory. You can get a detailed explanation of the function and usage of each module in some example (see page 16-26). We prepared an example of using three modules together.

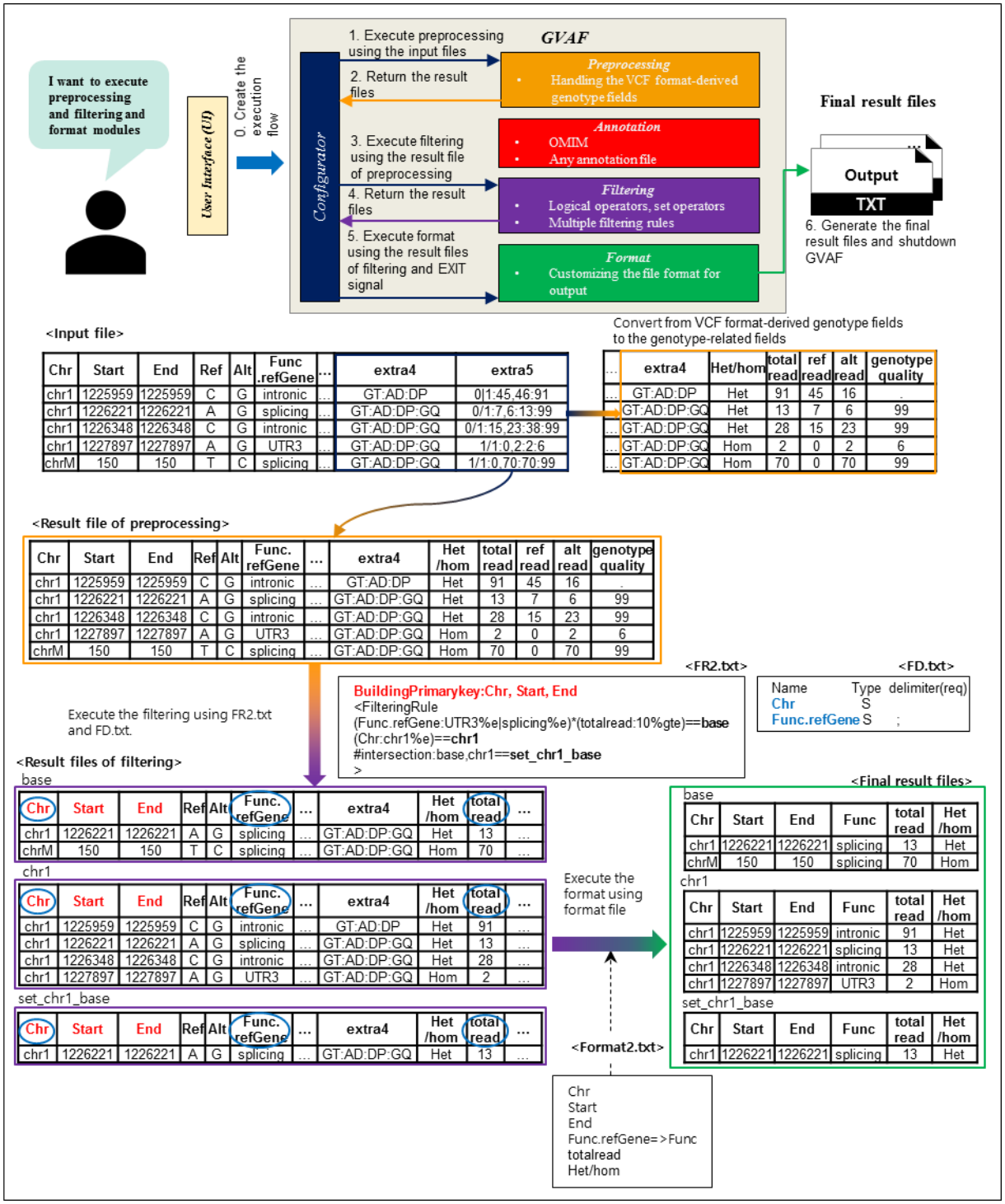


Figure 18. An example using the preprocessing and filtering and format modules.

Figure 18 shows the process of executing the preprocessing and filtering and format modules together. GVAF produces the execution flow in order of the preprocessing and filtering and format modules based on the predefined priority. The input file in this example is the Annovar-like output format, and the input file is used as input for the preprocessing module. The preprocessing module creates the result file by converting the VCF derived genotype data related fields in the input file into the user-friendly genotype fields. The result file of the preprocessing module is used as input

for the filtering module. The filtering module applies three filtering rules (i.e., base, chr1, set_chr1_base) in FR2.txt to the result file of the preprocessing module to generate the result file, and result file of the filtering module is used as input for the format module. The field information referenced in the filtering rules of FR2.txt is necessarily stored in FD.txt. However, the fields including the user-friendly genotype fields such as “het/hom, totalread, refread, altread, genotypequality” can be defined in the filtering rules even if they are not defined in FD.txt. For example, the “totalread” field is applied by the filtering rule with base alias and this field information is not defined in FD.txt. The format module exports 6 fields from the result file of the filtering module by referring Format2.txt and creates the result files consisting of exported fields.

You can execute an example in figure 17 using the following command.

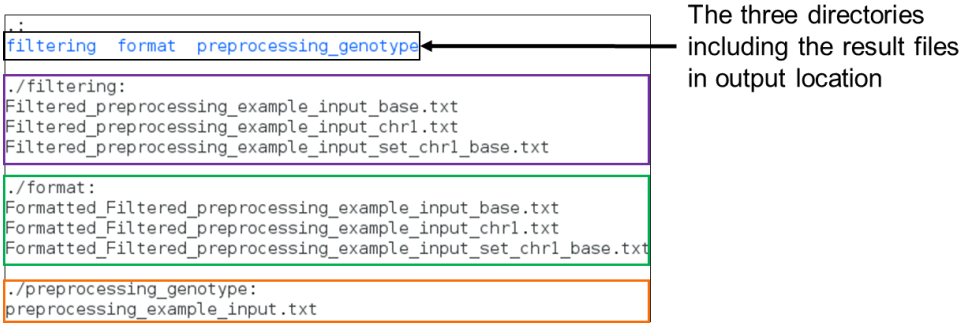
- Command for executing an example in figure 18

```
gvaf -input dummydata/example_input.txt -genotype_field "extra5" -genotype_format_field "extra4" -
FilteringRule_File FR2.txt -FieldDefinition_File FD.txt -format_file Format2.txt -outputdir result
```

Using this command, one file (named example_input.txt in dummydata directory) as input using “-input” option. The five options in preprocessing, filtering, format categories are set for executing several modules. Two options in preprocessing category such as "-genotype_field" and "-genotype_format_field" are set for executing the preprocessing module. The "-genotype_field" option is set by entering the field name (i.e., extra5) of input file including genotype-related data and the "-genotype_format_field" option is set by entering the field name (i.e., extra5) of input file including genotype-related data format. The two options in filtering category such as “-FilteringRule_File” and “–FieldDefinition_File” are set for executing the filtering module. The “-FilteringRule_File” option is set by entering the file path of FR2.txt as a FR file. The FieldDefinition_File” option is set by entering he file path of FD.txt as a FD file. One option in format category such as “-format_file” option is set for executing the format module. The “-format_file” option in the format category is set by entering the file path of Format2.txt as a format file. The result directory is set as output location for storing the result files using “-outputdir” option. All result files of each module are stored in individual directory of output directory as below.

You can find the final result files in format directory of the output location.

- The result files



gvaf_creation_annotation_file

The gvaf_creation_annotation_file tool is used to generate an annotation file.

Synopsis

gvaf_creation_annotation_file -input <file path> -outdir <directory path> -method <c/r> -field <field name> [-headertag STRING]

Options

These options are used to generate an annotation file that is no duplication in annotated field via the deduplication step.

-input <file path>

This option is used to specify input files.

-outdir <directory path>

This option is used to specify the output location.

-field <field name>

This option is used to specify field name for creating annotated field consisting of no duplicate values.

-method <c/r>

The option is used to specify deduplication method. GVAF currently provides two deduplication methods as described below.

- combine (c): combine duplicated columns into one.

- remove (r): remove duplicated columns.

- headertag <String>

This option appends headertag into all field names in a file.

Example

An example for generating an annotation file using the combine method.

e.g.,) gvaf_creation_annotation_file -input ~/workspace/GVAF/refdata/OMIM20160401.txt -outdir . -method c -headertag "OMIM." -field "Gene";

```
[skim@localhost scripts]$ gvaf_creation_annotation_file -input ~/workspace/GVAF/refdata/OMIM20160401.txt -outdir . -method c -headertag "OMIM." -field "Gene" ;
[Start]GVAFDBHandler
GVAFDB_configuration.method :combine
GVAFDBHandler time: 0.487sec.
[END]GVAFDBHandler
```

- The result using headertag “OMIM.”:

  <header of input> <header of output>
  “Gene, disease name, alternative titles” -> “OMIM.Gene, OMIM.disease name, OMIM.alternative titles”.