

Guide for trident v1.2.0.0 to v1.2.1.0

Contents

1	The trident CLI	1
1.1	General notes	3
1.1.1	Logging and command line output	3
1.1.2	Duplicates	3
1.1.3	Group names in .fam files	3
1.1.4	Whitespaces in the .janno file	4
2	Package creation and manipulation commands	4
2.1	Init command	4
2.2	Fetch command	5
2.3	Forge command	6
2.3.1	The forge selection language	8
2.3.2	Treatment of the .janno file while merging	9
2.3.3	Treatment of the .ssf file while merging	10
2.3.4	Other options	10
2.4	Genoconvert command	10
2.5	Update command	11
3	Inspection commands	13
3.1	List command	13
3.2	Summarise command	14
3.3	Survey command	14
3.4	Validate command	15

1 The trident CLI

Trident is a command line software tool structured in multiple subcommands. If you installed it properly you can call it on the command line by typing `trident`. This will show an overview of the general options and all subcommands, which are explained in detail below.

```
Usage: trident [--version] [--logMode ARG] [--errLength ARG]
           [--inPlinkPopName ARG] (COMMAND | COMMAND)
trident is a management and analysis tool for Poseidon packages. Report issues
here: https://github.com/poseidon-framework/poseidon-hs/issues
```

Available options:

<code>-h, --help</code>	Show this help text
<code>--version</code>	Show version number
<code>--logMode ARG</code>	How information should be reported: NoLog, SimpleLog, DefaultLog, ServerLog or VerboseLog (default: DefaultLog)
<code>--errLength ARG</code>	After how many characters should a potential error message be truncated. "Inf" for no truncation. (default: CharCount 1500)

```
--inPlinkPopName ARG      Where to read the population/group name from the FAM
                           file in Plink-format. Three options are possible:
                           asFamily (default) | asPhenotype | asBoth.
```

Package creation and manipulation commands:

```
init          Create a new Poseidon package from genotype data
fetch         Download data from a remote Poseidon repository
forge         Select packages, groups or individuals and create a
               new Poseidon package from them
genoconvert   Convert the genotype data in a Poseidon package to a
               different file format
update        Update POSEIDON.yml files automatically
```

Inspection commands:

```
list          List packages, groups or individuals from local or
               remote Poseidon repositories
summarise     Get an overview over the content of one or multiple
               Poseidon packages
summarize     Synonym for summarise
survey        Survey the degree of context information completeness
               for Poseidon packages
validate      Check one or multiple Poseidon packages for
               structural correctness
```

Trident allows to work directly with genotype data (see `-p` below), but its optimized for the interaction with [Poseidon packages](#), which wrap and contextualize the data. Most trident subcommands therefore have a central parameter, called `--baseDir` or simply `-d` to specify one or more base directories to look for packages. For example, if all Poseidon packages live inside a repository at `/path/to/poseidon/packages` you would simply say `trident <subcommand> -d /path/to/poseidon/dirs/` and `trident` would automatically search all subdirectories inside of the repository for valid Poseidon packages (as identified by valid `POSEIDON.yml` files).

You can arrange a poseidon repository in a hierarchical way. For example:

```
/path/to/poseidon/packages
  /modern
    /2019_poseidon_package1
    /2019_poseidon_package2
  /ancient
    /...
    /...
  /Reference_Genomes
    /...
    /...
```

You can use this structure to select only the level of packages you're interested in, even individual ones, and you can make use of the fact that `-d` can be given multiple times.

Being able to specify one or multiple repositories is often not enough, as you may have your own data to co-analyse with the main repository. This is easy to do, as you simply need to provide your own genotype data as yet another Poseidon package to be added to your `trident` command. For example, let's say you have genotype data in EIGENSTRAT format (`trident` supports EIGENSTRAT and PLINK as formats.):

```
~/my_project/my_project.geno
~/my_project/my_project.snp
~/my_project/my_project.ind
```

then you can make that to a skeleton Poseidon package with the `init` command. You can also do it manually by simply adding a `POSEIDON.yml` file, with for example the following content:

```
poseidonVersion: 2.5.0
title: My_awesome_project
```

```

description: Unpublished genetic data from my awesome project
contributor:
  - name: Stephan Schiffels
    email: schiffels@institute.org
packageVersion: 0.1.0
lastModified: 2020-10-07
genotypeData:
  format: EIGENSTRAT
  genoFile: my_project.geno
  snpFile: my_project.snp
  indFile: my_project.ind
jannoFile: my_project.janno
bibFile: sources.bib

```

Two remarks: 1) all file paths are considered *relative* to the directory in which `POSEIDON.yml` resides. Here we assume that you put this file into the same directory as the three genotype files. 2) Besides the genotype data files there are two (technically optional) files referenced by this example `POSEIDON.yml` file: `sources.bib` and `my_project.janno`. Of course you can add them manually - `init` automatically creates empty dummy versions.

Once you have set up your own “Poseidon” package (which is really only a skeleton so far), you can add it to your `trident` analysis, by simply adding your project directory to the command using `-d`, for example:

```

trident list -d /path/to/poseidon/packages/modern \
  -d /path/to/poseidon/packages/ReferenceGenomes
  -d ~/my_project --packages

```

1.1 General notes

1.1.1 Logging and command line output

For all subcommands the general argument `--logMode` defines how `trident` reports messages (to `stderr`) on the command line:

- *NoLog*: Hides all messages.
- *SimpleLog*: Plain and simple output to `stderr`.
- *DefaultLog*: Adds severity indicators before each message. (default setting)
- *ServerLog*: Additionally adds timestamps before each message.
- *VerboseLog*: Shows not just messages on the log levels **Info**, **Warning** and **Error** like the other modes, but also on the more verbose level **Debug**. Use this for debugging.

1.1.2 Duplicates

- If multiple packages in a package repository share the same **title**, then `trident` will try to select the one with the highest version number. If this is not sufficient to resolve the conflict, `trident` will stop. An exception for that is the `list` subcommand, which will read and report all packages/groups/individuals in all versions.
- Individual/sample names (`Poseidon_IDs`) within one package have to be unique, or `trident` will stop.
- We generally also discourage ID duplicates across packages in package repositories, but `trident` will generally continue with them after printing a warning. This does not apply for `validate`, by default (you can change this behaviour with `--ignoreDuplicates`), and `forge`. `forge` offers a special mechanism to resolve duplicates within its selection language (see below).

1.1.3 Group names in .fam files

The `.fam` file of Plink-formatted genotype data is used inconsistently across different popular aDNA software tools to store group/population name information. The (global) option `--inPlinkPopName` with the arguments `asFamily` (default), `asPhenotype` and `asBoth` allows to control the reading of the population name from Plink `.fam` files. The subcommands that write genotype data (`forge`, `genoconvert`) have a corresponding option `--outPlinkPopName` to specify this for the output.

1.1.4 Whitespaces in the .janno file

While reading the .janno file `trident` trims all leading and trailing whitespaces around individual cells. Also all instances of the `No-Break Space` unicode character will be removed. This means these whitespaces will not be preserved when a package is forged.

2 Package creation and manipulation commands

2.1 Init command

`init` creates a new, valid Poseidon package from genotype data files. It adds a valid `POSEIDON.yml` file, a dummy .janno file for context information and an empty .bib file for literature references.

[Click here for command line details](#)

```
Usage: trident init ((-p|--genoOne ARG) | --inFormat ARG --genoFile ARG
                  --snpFile ARG --indFile ARG) [--snpSet ARG]
                  (-o|--outPackagePath ARG) [-n|--outPackageName ARG]
                  [--minimal]
```

Create a new Poseidon package from genotype data

Available options:

<code>-h,--help</code>	Show this help text
<code>-p,--genoOne ARG</code>	one of the input genotype data files. Expects .bed or .bim or .fam for PLINK and .geno or .snp or .ind for EIGENSTRAT. The other files must be in the same directory and must have the same base name
<code>--inFormat ARG</code>	the format of the input genotype data: EIGENSTRAT or PLINK (only necessary for data input with --genoFile + --snpFile + --indFile)
<code>--genoFile ARG</code>	the input geno file path
<code>--snpFile ARG</code>	the input snp file path
<code>--indFile ARG</code>	the input ind file path
<code>--snpSet ARG</code>	the snpSet of the package: 1240K, HumanOrigins or Other. (only relevant for data input with -p --genoOne or --genoFile + --snpFile + --indFile, because the packages in a -d --baseDir already have this information in their respective POSEIDON.yml files) Default: Other
<code>-o,--outPackagePath ARG</code>	the output package directory path
<code>-n,--outPackageName ARG</code>	the output package name - this is optional: If no name is provided, then the package name defaults to the basename of the (mandatory) --outPackagePath argument
<code>--minimal</code>	should only a minimal output package be created?

The command

```
trident init \
  --inFormat EIGENSTRAT/PLINK \
  --genoFile path/to/geno_file \
  --snpFile path/to/snp_file \
  --indFile path/to/ind_file \
  --snpSet 1240K|HumanOrigins|Other \
  -o path/to/new_package_name
```

requires the format (`--inFormat`) of your input data (either `EIGENSTRAT` or `PLINK`), the paths to the respective files (`--genoFile`, `--snpFile`, `--indFile`), and optionally the “shape” of these files (`--snpSet`), so if they cover the 1240K, the `HumanOrigins` or an `Other` SNP set. A simpler interface added in `trident 0.29.0` is available with

-p (+ --snpSet).

	EIGENSTRAT	PLINK
genoFile	.geno	.bed
snpFile	.snp	.bim
indFile	.ind	.fam

The output package of `init` is created as a new directory `-o`, which should not already exist, and gets the package `title` corresponding to the basename of `-o`. You can also set the title explicitly with `-n`. The `--minimal` flag causes `init` to create a minimal package with a very basic POSEIDON.yml and no `.bib` and `.janno` files.

2.2 Fetch command

`fetch` allows to download Poseidon packages from a remote Poseidon server. Read more about this repository [here](#).

Click here for command line details

```
Usage: trident fetch (-d|--baseDir DIR)
        (--downloadAll |
        (--fetchFile ARG | (-f|--fetchString ARG)))
        [--remoteURL ARG]
```

Download data from a remote Poseidon repository

Available options:

<code>-h,--help</code>	Show this help text
<code>-d,--baseDir DIR</code>	a base directory to search for Poseidon Packages (could be a Poseidon repository)
<code>--downloadAll</code>	download all packages the server is offering
<code>--fetchFile ARG</code>	A file with a list of packages. Works just as <code>-f</code> , but multiple values can also be separated by newline, not just by comma. <code>-f</code> and <code>--fetchFile</code> can be combined.
<code>-f,--fetchString ARG</code>	List of packages to be downloaded from the remote server. Package names should be wrapped in asterisks: <code>*package_title*</code> . You can combine multiple values with comma, so for example: <code>"*package_1*, *package_2*, *package_3*"</code> . <code>fetchString</code> uses the same parser as <code>forgeString</code> , but does not allow excludes. If groups or individuals are specified, then packages which include these groups or individuals are included in the download.
<code>--remoteURL ARG</code>	URL of the remote Poseidon server (default: "https://server.poseidon-adna.org")

It works with

```
trident fetch -d ... -d ... \  
-f "*package_title_1*,*package_title_2*,*package_title_3*,group_name,<individual1>"
```

and the entities you want to download must be listed either in a simple string of comma-separated values, which can be passed via `-f/--fetchString`, or in a text file (`--fetchFile`). Entities are then combined from these sources.

Entities are specified using a special syntax (see also the documentation of `forge` below): Package titles are wrapped in asterisks: `*package_title*`, group names are spelled as is, and individual names are wrapped in angular brackets, so `<individual1>`. Fetch will figure out which packages need to be downloaded to include all specified entities. `--downloadAll`, which can be given instead of `-f` and `--fetchFile`, causes fetch to download

all packages from the server. The downloaded packages are added in the first (!) `-d` directory (which gets created if it doesn't exist), but downloads are only performed if the respective packages are not already present in the latest version in any of the `-d` dirs.

Note that `trident fetch` makes most sense in combination with `trident list --remote`: First one can inspect what is available on the server, then one can create a custom fetch command.

`fetch` also has the optional arguments `--remote https://..."` to name an alternative poseidon server. The default points to the [DAG server](#).

2.3 Forge command

`forge` creates new Poseidon packages by extracting and merging packages, populations and individuals from your Poseidon repositories.

[Click here for command line details](#)

```
Usage: trident forge ((-d|--baseDir DIR) |
    ((-p|--genoOne ARG) | --inFormat ARG --genoFile ARG
    --snpFile ARG --indFile ARG) [--snpSet ARG])
    [--forgeFile ARG | (-f|--forgeString ARG)]
    [--selectSnps ARG] [--intersect] [--outFormat ARG]
    [--minimal] [--onlyGeno] (-o|--outPackagePath ARG)
    [-n|--outPackageName ARG] [--packagewise]
    [--outPlinkPopName ARG]

Select packages, groups or individuals and create a new Poseidon package from
them
```

Available options:

<code>-h,--help</code>	Show this help text
<code>-d,--baseDir DIR</code>	a base directory to search for Poseidon Packages (could be a Poseidon repository)
<code>-p,--genoOne ARG</code>	one of the input genotype data files. Expects .bed or .bim or .fam for PLINK and .geno or .snp or .ind for EIGENSTRAT. The other files must be in the same directory and must have the same base name
<code>--inFormat ARG</code>	the format of the input genotype data: EIGENSTRAT or PLINK (only necessary for data input with --genoFile + --snpFile + --indFile)
<code>--genoFile ARG</code>	the input geno file path
<code>--snpFile ARG</code>	the input snp file path
<code>--indFile ARG</code>	the input ind file path
<code>--snpSet ARG</code>	the snpSet of the package: 1240K, HumanOrigins or Other. (only relevant for data input with -p --genoOne or --genoFile + --snpFile + --indFile, because the packages in a -d --baseDir already have this information in their respective POSEIDON.yml files) Default: Other
<code>--forgeFile ARG</code>	A file with a list of packages, groups or individual samples. Works just as -f, but multiple values can also be separated by newline, not just by comma. Empty lines are ignored and comments start with "#", so everything after "#" is ignored in one line. Multiple instances of -f and --forgeFile can be given. They will be evaluated according to their input order on the command line.
<code>-f,--forgeString ARG</code>	List of packages, groups or individual samples to be combined in the output package. Packages follow the syntax <code>*package_title*</code> , populations/groups are simply

group_id and individuals <individual_id>. You can combine multiple values with comma, so for example: "*package_1*, <individual_1>, <individual_2>, group_1". Duplicates are treated as one entry. Negative selection is possible by prepending "-" to the entity you want to exclude (e.g. "*package_1*, -<individual_1>, -group_1"). forge will apply excludes and includes in order. If the first entity is negative, then forge will assume you want to merge all individuals in the packages found in the baseDirs (except the ones explicitly excluded) before the exclude entities are applied. An empty forgeString (and no --forgeFile) will therefore merge all available individuals. If there are individuals in your input packages with equal individual id, but different main group or source package, they can be specified with the special syntax "<package:group:individual>".

--selectSnps ARG To extract specific SNPs during this forge operation, provide a Snp file. Can be either Eigenstrat (file ending must be '.snp') or Plink (file ending must be '.bim'). When this option is set, the output package will have exactly the SNPs listed in this file. Any SNP not listed in the file will be excluded. If option '--intersect' is also set, only the SNPs overlapping between the SNP file and the forged packages are output.

--intersect Whether to output the intersection of the genotype files to be forged. The default (if this option is not set) is to output the union of all SNPs, with genotypes defined as missing in those packages which do not have a SNP that is present in another package. With this option set, the forged dataset will typically have fewer SNPs, but less missingness.

--outFormat ARG the format of the output genotype data: EIGENSTRAT or PLINK. Default: PLINK

--minimal should only a minimal output package be created?

--onlyGeno should only the resulting genotype data be returned? This means the output will not be a Poseidon package

-o,--outPackagePath ARG the output package directory path

-n,--outPackageName ARG the output package name - this is optional: If no name is provided, then the package name defaults to the basename of the (mandatory) --outPackagePath argument

--packagewise Skip the within-package selection step in forge. This will result in outputting all individuals in the relevant packages, and hence a superset of the requested individuals/groups. It may result in better performance in cases where one wants to forge entire packages or almost entire packages. Details: Forge conceptually performs two types of selection: First, it identifies which packages in the supplied base directories are relevant to the requested forge, i.e. whether they are either explicitly listed using *PackageName*, or because they contain selected individuals or groups. Second, within each relevant package, individuals which are not requested are

removed. This option skips only the second step, but still performs the first.

`--outPlinkPopName ARG` Where to write the population/group name into the FAM file in Plink-format. Three options are possible: `asFamily` (default) | `asPhenotype` | `asBoth`. See also `--inPlinkPopName`.

`forge` can be used with

```
trident forge -d ... -d ... \
  -f "*package_name*, group_id, <individual_id>" \
  -o path/to/new_package_name
```

where the entities (packages, groups/populations, individuals/samples) you want in the output package can be denoted either as a string on the command line (`-f/--forgeString`), or in an input text file (`--forgeFile`). See the section below for the syntax of this selection language. Do not forget to wrap the `--forgeString` query in quotes.

Including one or multiple Poseidon packages with `-d` is not the only way to include data for a `forge` operation. It is also possible to consider unpackaged genotype data directly with `-p` (+ `--snpSet`) or `--inFormat` + `--genoFile` + `--snpFile` + `--indFile` (+ `--snpSet`). This makes the following example possible, where we merge data from one Poseidon package and two genotype datasets to get a new EIGENSTRAT dataset.

```
trident forge \
  -d 2017_GonzalesFortesCurrentBiology \
  -p 2018_VeeramahPNAS/2018_VeeramahPNAS.fam \
  --inFormat PLINK \
  --genoFile 2017_HaberAJHG/2017_HaberAJHG.bed \
  --snpFile 2017_HaberAJHG/2017_HaberAJHG.bim \
  --indFile 2017_HaberAJHG/2017_HaberAJHG.fam \
  -f "<STR241.SG>,<ERS1790729.SG>,Iberia_HG.SG" \
  -o testpackage \
  --outFormat EIGENSTRAT \
  --onlyGeno
```

2.3.1 The forge selection language

The text in `--forgeString` and `--forgeFile` are parsed as a domain specific query language that describes precisely which entities should be compiled in the output package of a given `forge` operation. The language has multiple syntactic elements and a specific evaluation logic.

In general a `--forgeString` query consists of multiple entities, separated by `,`. The main entities are Poseidon packages, groups/populations and individuals/samples:

- Each package title is surrounded by `*`: `*package*`. That means if you want all individuals of the Poseidon package `2019_Jeong_InnerEurasia` in the output package you would add `*2019_Jeong_InnerEurasia*` to the query.
- Groups/populations are not specially marked: `group`. So to get all individuals of the group `Swiss_Roman_period`, you would simply add `Swiss_Roman_period`.
- Individuals/samples are surrounded by `<` and `>`: `<individual>`. `ALA026` therefore becomes `<ALA026>`. A second way to denote individuals is with the more verbose and specific syntax `<package:group:individual>`. Such defined individuals take precedence over differently defined ones (so: directly with `<individual>` or as a subset of `*package*` or `group`). This allows to resolve duplication issues precisely – at least in cases where the duplicated individuals differ in source package or primary group.

In the `--forgeFile` each line is treated as a separate `forgeString`, empty lines are ignored and `#`s start comments. So this is a valid `forgeFile`:

```
# Packages
*package1*, *package2*
```



```
# Groups and individuals from other packages beyond package1 and package2
group1, <individual1>, group2, <individual2>, <individual3>

# group2 has two outlier individuals that should be ignored
-<bad_individual1> # This one has very low coverage
-<bad_individual2> # This one is from a different time period
```

By prepending - to the bad individuals, we can exclude them from the forged package. `forge` figures out the final list of samples to include by executing all forge-entities in order. So an entity list `*PackageA*,-<Individual1>,GroupA` may result in a different outcome than `*PackageA*,GroupA,-<Individual1>`, depending on whether `<Individual1>` belongs to `GroupA` or not. If the forge entity list starts with a negative entity, or if the entity list is empty, `forge` will implicitly assume you want to include all individuals in all packages found in the `baseDirs` (except the ones explicitly excluded, of course).

An empty `forgeString` will therefore merge all available individuals.

2.3.2 Treatment of the .janno file while merging

`forge` merges and subsets .janno files along with the genotype data. If a package lacks a .janno file, then a basic one will be created internally based on the information in the genotype data, and used for the output. Missing columns across packages will be filled with `n/a`.

For merging two .janno files **A** and **B** the following rules apply regarding undefined, arbitrary additional columns:

- If **A** has an additional column which is not in **B** then empty cells in the rows imported from **B** are filled with `n/a`.
- If **A** and **B** share additional columns with identical column name, then they are treated as semantically identical units and merged accordingly.
- In the resulting .janno file, all additional columns from both **A** and **B** are sorted alphabetically and appended after the normal, specified variables.

The following example illustrates the described behaviour:

A.janno

Poseidon_ID	Group_Name	Genetic_Sex	AdditionalColumn1	AdditionalColumn2
XXX011	POP1	M	A	D
XXX012	POP2	F	B	E
XXX013	POP1	M	C	F

B.janno

Poseidon_ID	Group_Name	Genetic_Sex	AdditionalColumn3	AdditionalColumn2
YYY022	POP5	F	G	J
YYY023	POP5	F	H	K
YYY024	POP5	M	I	L

A.janno + B.janno

Poseidon_ID	Group_Name	Genetic_Sex	AdditionalColumn1	AdditionalColumn2	AdditionalColumn3
XXX011	POP1	M	A	D	n/a
XXX012	POP2	F	B	E	n/a
XXX013	POP1	M	C	F	n/a
YYY022	POP5	F	n/a	J	G
YYY023	POP5	F	n/a	K	H
YYY024	POP5	M	n/a	L	I

2.3.3 Treatment of the .ssf file while merging

The Sequencing Source File (short .ssf file) is forged in exactly the same way as the janno file. SSF files that are present are included in the forge product in the way that the user expects, following selection of those entities which are listed in the `poseidon_IDs` columns of the SSF files. Columns that are only present in some packages, including those not defined by our [Schema] are also included in the forged product in the same way as described for Janno above.

2.3.4 Other options

Just as for `init` the output package of `forge` is created as a new directory `-o`. The title can also be explicitly defined with `-n`.

`--minimal` allows for the creation of a minimal output package without `.bib` and `.janno`. This is especially useful for data analysis pipelines, where only the genotype data is required. Even more basic output comes with `--onlyGeno`, which means that only the genotype data is returned without any Poseidon package.

`forge` has a an optional flag `--intersect`, that defines, if the genotype data from different packages should be merged with an **union** or an **intersect** operation. The default (if this option is not set) is to output the union of all SNPs, with genotypes defined as missing in samples from packages which do not have a SNP that is present in another package. With this option set, on the other hand, the forged dataset will typically have fewer SNPs, but less missingness.

`--intersect` also influences the automatic determination of the `snpSet` field in the POSEIDON.yml file for the resulting package. If the `snpSets` of all input packages are identical, then the resulting package will just inherit this configuration. Otherwise `forge` applies the following pairwise merging logic:

Input snpSet A	Input snpSet B	--intersect	Ouput snpSet
Other	*	*	Other
1240K	HumanOrigins	True	HumanOrigins
1240K	HumanOrigins	False	1240K

`--selectSnps` allows to provide `forge` with a SNP file in EIGENSTRAT (`.snp`) or PLINK (`.bim`) format to create a package with a specific selection. When this option is set, the output package will have exactly the SNPs listed in this file. Any SNP not listed in the file will be excluded. If `--intersect` is also set, only the SNPs overlapping between the SNP file and the forged packages are output.

Merging genotype data across different data sources and file formats is tricky. `forge` is more verbose about potential issues, if the `--logMode` flag is set to `VerboseLog`.

The `--onlyGeno` command specifies that only genotype data should be output, not an entire Poseidon package.

With `--packagewise` the within-package selection step in `forge` can be skipped. This will result in outputting all individuals in the relevant packages, and hence a superset of the requested individuals/groups. It may result in better performance in cases where one wants to forge entire packages.

2.4 Genoconvert command

`genoconvert` converts the genotype data in a Poseidon package to a different file format. The respective entries in the POSEIDON.yml file are changed accordingly.

[Click here for command line details](#)

```
Usage: trident genoconvert ((-d|--baseDir DIR) |
    ((-p|--genoOne ARG) | --inFormat ARG --genoFile ARG
    --snpFile ARG --indFile ARG) [--snpSet ARG])
    --outFormat ARG [--onlyGeno]
    [-o|--outPackagePath ARG] [--removeOld]
    [--outPlinkPopName ARG]
```

Convert the genotype data in a Poseidon package to a different file format

Available options:

<code>-h,--help</code>	Show this help text
<code>-d,--baseDir DIR</code>	a base directory to search for Poseidon Packages (could be a Poseidon repository)
<code>-p,--genoOne ARG</code>	one of the input genotype data files. Expects .bed or .bim or .fam for PLINK and .geno or .snp or .ind for EIGENSTRAT. The other files must be in the same directory and must have the same base name
<code>--inFormat ARG</code>	the format of the input genotype data: EIGENSTRAT or PLINK (only necessary for data input with --genoFile + --snpFile + --indFile)
<code>--genoFile ARG</code>	the input geno file path
<code>--snpFile ARG</code>	the input snp file path
<code>--indFile ARG</code>	the input ind file path
<code>--snpSet ARG</code>	the snpSet of the package: 1240K, HumanOrigins or Other. (only relevant for data input with -p --genoOne or --genoFile + --snpFile + --indFile, because the packages in a -d --baseDir already have this information in their respective POSEIDON.yml files) Default: Other
<code>--outFormat ARG</code>	the format of the output genotype data: EIGENSTRAT or PLINK.
<code>--onlyGeno</code>	should only the resulting genotype data be returned? This means the output will not be a Poseidon package
<code>-o,--outPackagePath ARG</code>	the output package directory path - this is optional: If no path is provided, then the output is written to the directories where the input genotype data file (.bed/.geno) is stored
<code>--removeOld</code>	Remove the old genotype files when creating the new ones
<code>--outPlinkPopName ARG</code>	Where to write the population/group name into the FAM file in Plink-format. Three options are possible: asFamily (default) asPhenotype asBoth. See also --inPlinkPopName.

With the default setting

```
trident genoconvert -d ... -d ... --outFormat EIGENSTRAT|PLINK
```

all packages in -d will be converted to the desired --outFormat (either EIGENSTRAT or PLINK), if the data is not already in this format. This includes updating the respective POSEIDON.yml files.

The “old” data is not deleted, but kept around. That means conversion can result in a package with both PLINK and EIGENSTRAT data, but only one is linked in the POSEIDON.yml file, and that is what will be used by trident. To delete the old data in the conversion you can add the --removeOld flag.

Instead of -d to change Poseidon packages, the -p (+ --snpSet) or --inFormat + --genoFile + --snpFile + --indFile (+ --snpSet) allow to directly convert genotype data that is not wrapped in a Poseidon package and store it to a directory given in -o. See this example:

```
trident genoconvert \  
-p 2018_Mittnik_Baltic/Mittnik_Baltic.bed \  
--outFormat EIGENSTRAT \  
-o my_directory
```

2.5 Update command

update automatically harmonizes POSEIDON.yml files of one or multiple packages if the packages were changed. This is not an automatic update from one Poseidon version to the next!

Click here for command line details

```
Usage: trident update (-d|--baseDir DIR) [--poseidonVersion ARG]
      [--ignorePoseidonVersion] [--versionComponent ARG]
      [--noChecksumUpdate] [--newContributors ARG]
      [--logText ARG] [--force]
```

Update POSEIDON.yml files automatically

Available options:

<code>-h,--help</code>	Show this help text
<code>-d,--baseDir DIR</code>	a base directory to search for Poseidon Packages (could be a Poseidon repository)
<code>--poseidonVersion ARG</code>	Poseidon version the packages should be updated to: e.g. "2.5.3" (default: Nothing)
<code>--ignorePoseidonVersion</code>	Read packages even if their poseidonVersion is not compatible with the trident version. The assumption is, that the package is already structurally adjusted to the trident version and only the version number is lagging behind.
<code>--versionComponent ARG</code>	Part of the package version number in the POSEIDON.yml file that should be updated: Major, Minor or Patch (see https://semver.org) (default: Patch)
<code>--noChecksumUpdate</code>	Should update of checksums in the POSEIDON.yml file be skipped
<code>--ignoreGeno</code>	ignore SNP and GenoFile
<code>--newContributors ARG</code>	Contributors to add to the POSEIDON.yml file in the form "[Firstname Lastname](Email address);..."
<code>--logText ARG</code>	Log text for this version jump in the CHANGELOG file (default: "not specified")
<code>--force</code>	Normally the POSEIDON.yml files are only changed if the poseidonVersion is adjusted or any of the checksums change. With --force a package version update can be triggered even if this is not the case.

It can be called with a lot of optional arguments

```
trident update -d ... -d ... \
  --poseidonVersion "X.X.X" \
  --versionComponent Major/Minor/Patch \
  --noChecksumUpdate
  --ignoreGeno
  --newContributors "[Firstname Lastname](Email address);..."
  --logText "short description of the update"
  --force
```

By default `update` will not edit a package's POSEIDON.yml file, even when arguments like `--versionComponent`, `--newContributors` or `--logText` are explicitly set. This default exists to run the function on a large set of packages where only few of them were edited and need an active update. A package will only be modified by `update` if either

- any of the files with checksums (e.g. the genotype data) in it were modified,
- the `--poseidonVersion` argument differs from the `poseidonVersion` in the package's POSEIDON.yml file
- or the `--force` flag was set in `update`.

If any of these applies to a package in the search directory (`--baseDir/-d`), it will be updated. This includes the following steps:

- If `--poseidonVersion` is different from the `poseidonVersion` field in the package, then that will be

updated.

- The `packageVersion` will be incremented. If `--versionComponent` is not set, then it falls back to `Patch`, so a change in the last position of the three digit version number. Minor increments the middle, and Major the first position (see [semantic versioning](#)).
- The `lastModified` field will be updated to the current day (based on your computer's system time).
- The contributors in `--newContributors` will be added to the `contributor` field if they're not there already.
- If any checksums changed, then they will be updated. If certain checksums are not set yet, then they will be added. The checksum update can be skipped with `--noChecksumUpdate` or partially skipped for the genotype data with `--ignoreGeno`.
- The `CHANGELOG.md` file will be updated with a new row for the new version and the text in `--logText` (default: "not specified"), which will be appended as the first line of the file. If no `CHANGELOG.md` file exists, then it will be created and referenced in the `POSEIDON.yml` file.

`:heavy_exclamation_mark:` As `update` reads and rewrites `POSEIDON.yml` files, it may change their inner order, layout or even content (e.g. if they have fields which are not in the [Poseidon package definition](#)). Create a backup of the `POSEIDON.yml` file before running `update` if you are uncertain.

3 Inspection commands

3.1 List command

`list` lists packages, groups and individuals of the datasets you use, or of the packages available on the server.

[Click here for command line details](#)

```
Usage: trident list ((-d|--baseDir DIR) | --remote [--remoteURL ARG])
                (--packages | --groups | --individuals
                [-j|--jannoColumn JANNO_HEADER]) [--raw]
```

List packages, groups or individuals from local or remote Poseidon repositories

Available options:

<code>-h,--help</code>	Show this help text
<code>-d,--baseDir DIR</code>	a base directory to search for Poseidon Packages (could be a Poseidon repository)
<code>--remote</code>	list packages from a remote server instead the local file system
<code>--remoteURL ARG</code>	URL of the remote Poseidon server (default: "https://server.poseidon-adna.org")
<code>--packages</code>	list all packages
<code>--groups</code>	list all groups, ignoring any group names after the first as specified in the Janno-file
<code>--individuals</code>	list individuals
<code>-j,--jannoColumn JANNO_HEADER</code>	list additional fields from the janno files, using the Janno column heading name, such as Country, Site, Date_C14_Uncal_BP, Endogenous, ...
<code>--raw</code>	output table as tsv without header. Useful for piping into grep or awk

To list packages from your local repositories, as seen above you can run

```
trident list -d ... -d ... --packages
```

This will yield a nicely formatted table of all packages, their last update and the number of individuals in it.

To view packages on the remote server, instead of using directories to specify the locations of repositories on your system, you can use `--remote` to show packages on the remote server. For example

```
trident list --packages --remote
```

will result in a view of all published packages in our [public online repository](#).

You can also list groups, as defined in the third column of EIGENSTRAT .ind files (or the first/last column of a PLINK .fam file), and individuals with --groups and --individuals instead of --packages.

The --individuals flag provides a way to immediately access information from the .janno files on the command line. This works with the -j/--jannoColumn option. For example adding -j Country -j Date_C14_Uncal_BP to the commands above will add the Country and the Date_C14_Uncal_BP columns to the respective output tables.

Note that if you want a less fancy table, for example because you want to load this into Excel, or pipe into another command that cannot deal with the neat table layout, you can use the --raw option to output that table as a simple tab-delimited stream.

3.2 Summarise command

summarise prints some general summary statistics for a given poseidon dataset taken from the .janno files.

[Click here for command line details](#)

Usage: trident summarise (-d|--baseDir DIR) [--raw]

Get an overview over the content of one or multiple Poseidon packages

Available options:

-h,--help	Show this help text
-d,--baseDir DIR	a base directory to search for Poseidon Packages (could be a Poseidon repository)
--raw	output table as tsv without header. Useful for piping into grep or awk

You can run it with

```
trident summarise -d ... -d ...
```

which will show you context information like – among others – the number of individuals in the dataset, their sex distribution, the mean age of the samples (for ancient data) or the mean coverage on the 1240K SNP array in a table. summarise depends on complete .janno files and will silently ignore missing information for some statistics.

You can use the --raw option to output the summary table in a simple, tab-delimited layout.

3.3 Survey command

survey tries to indicate package completeness (mostly focused on .janno files) for poseidon datasets.

[Click here for command line details](#)

Usage: trident survey (-d|--baseDir DIR) [--raw]

Survey the degree of context information completeness for Poseidon packages

Available options:

-h,--help	Show this help text
-d,--baseDir DIR	a base directory to search for Poseidon Packages (could be a Poseidon repository)
--raw	output table as tsv without header. Useful for piping into grep or awk

Running

```
trident survey -d ... -d ...
```

will yield a table with one row for each package. See `trident survey -h` for a legend which cell of this table means what.

Again you can use the `--raw` option to output the survey table in a tab-delimited format.

3.4 Validate command

`validate` checks poseidon datasets for structural correctness.

Click [here](#) for command line details

Usage: `trident validate (-d|--baseDir DIR)`

Check one or multiple Poseidon packages for structural correctness

Available options:

<code>-h,--help</code>	Show this help text
<code>-d,--baseDir DIR</code>	a base directory to search for Poseidon Packages (could be a Poseidon repository)
<code>--ignoreGeno</code>	ignore SNP and GenoFile
<code>--fullGeno</code>	test parsing of all SNPs (by default only the first 100 SNPs are probed)
<code>--noExitCode</code>	do not produce an explicit exit code
<code>--ignoreDuplicates</code>	do not stop on duplicated individual names in the package collection

You can run it with

```
trident validate -d ... -d ...
```

and it will either report a success (**Validation passed**) or failure with specific error messages to simplify fixing the issues.

`validate` tries to ensure that each package in the dataset adheres to the [schema definition](#). Here is a list of what is checked:

- Presence of the necessary files
- Full structural correctness of `.bib` and `.janno` file
- Superficial correctness of genotype data files by parsing the first 100 SNPs. A full check that parses all SNPs can be run with the `--fullGeno` option
- Correspondence of BibTeX keys in `.bib` and `.janno`
- Correspondence of individual and group IDs in `.janno` and genotype data files

In fact much of this validation already runs as part of the general package reading pipeline invoked for many `trident` subcommands (e.g. `forge`). `validate` is meant to be more thorough, though, and will explicitly fail if even a single package is broken.

Remember to run it with `--logMode VerboseLog` to get more information if the output is not sufficient to debug an issue.