Contents

0.1 Guide for trident v0.29.0

0.1.1 Poseidon package repositories

Trident generally requires Poseidon “packages” to work with (since version 0.28.0 it also supports direct interaction with “unpackaged” genotype data – see `-p` below). 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
        /...
        /...
    /Archaic_Humans
        /...
        /...
```

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

Let’s use the `list` command to list all packages in the `modern` and `Reference_Genomes`:

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

0.1.2 Analysing your own dataset outside of the main repository

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 list` 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description: Unpublished genetic data from my awesome project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 I 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`:

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

0.1.3 Package creation and manipulation commands

0.1.3.1 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) | (-r|--inFormat ARG)
                      (-g|--genoFile ARG) (-s|--snpFile ARG) (-i|--indFile ARG))
                    [--snpSet ARG] (-o|--outPackagePath ARG)
                    [-n|--outPackageName ARG] [--minimal]
  Create a new Poseidon package from genotype data

Available options:
  -h,--help                Show this help text
  -p,--genoOne ARG         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
  -r,--inFormat ARG          the format of the input genotype data: EIGENSTRAT or
                             PLINK
  -g,--genoFile ARG          the input geno file path
  -s,--snpFile ARG           the input snp file path
  -i,--indFile ARG           the input ind file path
  --snpSet ARG               the snpSet of the new package: 1240K, HumanOrigins or
                             Other. Default: Other
  -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
  --minimal                  should only a minimal output package be created?
```

The command

```
trident init \
  -r EIGENSTRAT/PLINK \
  -g path/to/geno_file \
  -s path/to/snp_file \
  -i path/to/ind_file \
  --snpSet 1240K|HumanOrigins|Other \
  -o path/to/new_package_name
```

requires the format `-r` (`--inFormat`) of your input data (either `EIGENSTRAT` or `PLINK`), the paths to the respective files in `-g` (`--genoFile`), `-s` (`--snpFile`), and `-i` (`--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.

0.1.3.2 Fetch command `fetch` allows to download poseidon packages from a remote poseidon server.

Click here for command line details

```
Usage: trident fetch (-d|--baseDir DIR) [-f|--fetchString ARG] [--fetchFile ARG]
                     [--remoteURL ARG] [-u|--upgrade] [--downloadAll]
  Download data from a remote Poseidon repository
```

```

Available options:
  -h,--help                Show this help text
  -d,--baseDir DIR         a base directory to search for Poseidon Packages
                           (could be a Poseidon repository)
  -f,--fetchString ARG     List of packages to be downloaded from the remote
                           server. Package names should be wrapped in asterisks:
                           *package_title*. You can combine multiple values with
                           comma, so for example: "*package_1*, *package_2*,
                           *package_3*". fetchString uses the same parser as
                           forgeString, but does not allow excludes. If groups
                           or individuals are specified, then packages which
                           include these groups or individuals are included in
                           the download.
  --fetchFile ARG          A file with a list of packages. Works just as -f, but
                           multiple values can also be separated by newline, not
                           just by comma. -f and --fetchFile can be combined.
  --remoteURL ARG          URL of the remote Poseidon
                           server (default: "https://c107-224.cloud.gwdg.de")
  -u,--upgrade             overwrite outdated local package versions
  --downloadAll            download all packages the server is offering
```

It works with

```
trident fetch -d ... -d ... \
  -f "*package_title_1*,*package_title_2*,*package_title_3*,group_name,<Individual1>" \
  --fetchFile path/to/forgeFile
```

and the entities you want to download must be listed either in a simple string with comma-separated values (`-f`/`--fetchString`) or in a text file (`--fetchFile`). Entities are specified using a special syntax: Package titles are wrapped in asterisks: *package_title* (see also the documentation of `forge` below), group names are spelled as is, and individual names are wrapped in angular brackets, liks `<Individual1>`. Fetch will figure out which packages need to be downloaded to include all specified entities. `--downloadAll` causes fetch to ignore `-f` and download all packages from the server. The downloaded packages are added in the first (!) `-d` directory, but downloads are only performed if the respective packages are not already present in an up-to-date 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:://..."` do name an alternative poseidon server. The default points to the DAG server.

To overwrite outdated package versions with `fetch`, the `-u`/`--upgrade` flag has to be set. Note that many file systems do not offer a way to recover overwritten files. So be careful with this switch.

0.1.3.3 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) | (-r|--inFormat ARG)
                         (-g|--genoFile ARG) (-s|--snpFile ARG)
                         (-i|--indFile ARG)) [--snpSet ARG]]
                     [--forgeFile ARG | (-f|--forgeString ARG)]
                     [--selectSnps ARG] [--intersect] [--outFormat ARG]
                     [--minimal] [--onlyGeno] (-o|--outPackagePath ARG)
                     [-n|--outPackageName ARG] [-w|--warnings] [--no-extract]
  Select packages, groups or individuals and create a new Poseidon package from
  them

Available options:
  -h,--help                Show this help text
  -d,--baseDir DIR         a base directory to search for Poseidon Packages
                           (could be a Poseidon repository)
  -p,--genoOne ARG         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
  -r,--inFormat ARG        the format of the input genotype data: EIGENSTRAT or
                           PLINK
  -g,--genoFile ARG        the input geno file path
  -s,--snpFile ARG         the input snp file path
  -i,--indFile ARG         the input ind file path
  --snpSet ARG             the snpSet of the new package: 1240K, HumanOrigins or
                           Other. Default: Other
  --forgeFile ARG          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.
  -f,--forgeString ARG     List of packages, groups or individual samples to be
                           combined in the output package. Packages follow the
                           syntax *package_title*, 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
                          will therefore merge all available individuals.
  --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
  -w,--warnings           Show all warnings for merging genotype data
  --no-extract            Skip the 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. Note that this
                          will also ignore any ordering in the output
                          groups/individuals. With this option active,
                          individuals from the relevant packages will just be
                          written in the order that they appear in the original
                          packages.
```

`forge` can be used with

```
trident forge -d ... -d ... \
  -f "*package_name*, group_id, <individual_id>" \
  --forgeFile path/to/forgeFile \
  -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 as simple string with comma-separated values (`-f`/`--forgeString`) or in a text file (`--forgeFile`).

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 include unpackaged genotype data directly with `-r + -g + -s + -i (+ --snpSet)` or `-p (+ --snpSet)`. This makes the following example possible, where we merge data from one Poseidon package and two genotype datasets.

```
trident forge \
  -d 2017_GonzalesFortesCurrentBiology \
  -r PLINK \
  -g 2017_HaberAJHG/2017_HaberAJHG.bed \
  -s 2017_HaberAJHG/2017_HaberAJHG.bim \
  -i 2017_HaberAJHG/2017_HaberAJHG.fam \
  -r PLINK \
  -g 2018_VeeramahPNAS/2018_VeeramahPNAS.bed \
  -i 2018_VeeramahPNAS/2018_VeeramahPNAS.fam \
  -s 2018_VeeramahPNAS/2018_VeeramahPNAS.bim \
  -f "<STR241.SG>,<ERS1790729.SG>,Iberia_HG.SG" \
  -o testpackage \
  --onlyGeno
```

0.1.3.3.1 The forge selection language Entities in the `--forgeString` or the `--forgeFile` have to be marked in a certain way:

- Each package is surrounded by `*`, so if you want all individuals of `2019_Jeong_InnerEurasia` in the output package you would add `*2019_Jeong_InnerEurasia*` to the list.
- Groups/populations are not specially marked. So to get all individuals of the group `Swiss_Roman_period`, you would simply add `Swiss_Roman_period`.
- Individuals/samples are surrounded by `<` and `>`, so `ALA026` becomes `<ALA026>`.

Do not forget to wrap the forgeString in quotes.

You can either use `-f`/`--forgeString` or `--forgeFile`. In the file 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.

0.1.3.3.2 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 might be 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 `snpSet`s 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 `-w`/`--warnings` flag is set.

0.1.3.4 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) | (-r|--inFormat ARG)
                               (-g|--genoFile ARG) (-s|--snpFile ARG)
                               (-i|--indFile ARG)) [--snpSet ARG]]
                           --outFormat ARG [--onlyGeno]
```

```
                         [-o|--outPackagePath ARG] [--removeOld]
  Convert the genotype data in a Poseidon package to a different file format

Available options:
  -h,--help                Show this help text
  -d,--baseDir DIR         a base directory to search for Poseidon Packages
                           (could be a Poseidon repository)
  -p,--genoOne ARG         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
  -r,--inFormat ARG        the format of the input genotype data: EIGENSTRAT or
                           PLINK
  -g,--genoFile ARG        the input geno file path
  -s,--snpFile ARG         the input snp file path
  -i,--indFile ARG         the input ind file path
  --snpSet ARG             the snpSet of the new package: 1240K, HumanOrigins or
                           Other. Default: Other
  --outFormat ARG          the format of the output genotype data: EIGENSTRAT or
                           PLINK.
  --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 - 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
  --removeOld              Remove the old genotype files when creating the new
                           ones
```

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 combination `-r + -g + -s + -i (+ --snpSet)` or `-p (+ --snpSet)` allows 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
```

0.1.3.5 Update command `update` automatically updates POSEIDON.yml files of one or multiple packages if the packages were changed.

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:
  -h,--help                Show this help text
  -d,--baseDir DIR         a base directory to search for Poseidon Packages
                           (could be a Poseidon repository)
  --poseidonVersion ARG    Poseidon version the packages should be updated to:
                           e.g. "2.5.3" (default: Nothing)
  --ignorePoseidonVersion  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.
  --versionComponent ARG   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)
  --noChecksumUpdate       Should update of checksums in the POSEIDON.yml file
                           be skipped
  --ignoreGeno             ignore SNP and GenoFile
  --newContributors ARG    Contributors to add to the POSEIDON.yml file in the
                           form "[Firstname Lastname](Email address);..."
  --logText ARG            Log text for this version jump in the CHANGELOG
                           file (default: "not specified")
  --force                  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.

0.1.4 Inspection commands

0.1.4.1 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:
  -h,--help                Show this help text
  -d,--baseDir DIR         a base directory to search for Poseidon Packages
                           (could be a Poseidon repository)
  --remote                 list packages from a remote server instead the local
                           file system
  --remoteURL ARG          URL of the remote Poseidon
                           server (default: "https://c107-224.cloud.gwdg.de")
  --packages               list all packages
  --groups                 list all groups, ignoring any group names after the
                           first as specified in the Janno-file
  --individuals            list individuals
  -j,--jannoColumn JANNO_HEADER
                           list additional fields from the janno files, using
                           the Janno column heading name, such as Country, Site,
                           Date_C14_Uncal_BP, Endogenous, ...
  --raw                    output table as tsv without header. Useful for piping
                           into grep or awk
  --ignoreGeno             ignore SNP and GenoFile
```

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

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

This will yield a table like this

```
.-----------------------------------------.------------.-----------------.
|                  Title                  |    Date    | Nr Individuals |
:=========================================:============:=================:
| 2015_1000Genomes_1240K_haploid_pulldown | 2020-08-10 | 2535            |
| 2016_Mallick_SGDP1240K_diploid_pulldown | 2020-08-10 | 280             |
| 2018_BostonDatashare_modern_published   | 2020-08-10 | 2772            |
| ...                                     | ...        |                 |
'-----------------------------------------'------------'-----------------'
```

so 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 column of a PLINK `.fam` file), and individuals:

```
trident list -d ... -d ... --groups
trident list -d ... -d ... --individuals
```

The `--individuals` flag also 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 `--jannoColum Country --jannoColumn 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.

0.1.4.2 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.

0.1.4.3 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.

0.1.4.4 Validate command `validate` checks poseidon datasets for structural correctness.

Click here for command line details

```
Usage: trident validate (-d|--baseDir DIR) [--verbose]
  Check one or multiple Poseidon packages for structural correctness

Available options:
  -h,--help                Show this help text
  -d,--baseDir DIR         a base directory to search for Poseidon Packages
                           (could be a Poseidon repository)
  --verbose                print more output to the command line
  --ignoreGeno             ignore SNP and GenoFile
  --noExitCode             do not produce an explicit exit code
```

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. A full check would be too computationally expensive
- 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.