# Package ‘genio’

December 18, 2019

**Title** Genetics Input/Output Functions

**Version** 1.0.12

**Description** Implements readers and writers for file formats associated with genetics data. Reading and writing plink BED/BIM/FAM formats is fully supported, including a lightning-fast BED reader and writer implementations. Other functions are 'readr' wrappers that are more constrained, user-friendly, and efficient for these particular applications; handles plink and eigenstrat tables (FAM, BIM, IND, and SNP files). There are also ``make'' functions for FAM and BIM tables with default values to go with simulated genotype data.

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.0.2

**Imports** readr, tibble, Rcpp

**LinkingTo** Rcpp

**Suggests** testthat, knitr, rmarkdown, lfa, BEDMatrix, snpStats, pryr

**VignetteBuilder** knitr

**URL** `https://github.com/OchoaLab/genio`

**BugReports** `https://github.com/OchoaLab/genio/issues`

**NeedsCompilation** yes

**Author** Alejandro Ochoa [aut, cre] (<https://orcid.org/0000-0003-4928-3403>)

**Maintainer** Alejandro Ochoa <`alejandro.ochoa@duke.edu`>

**Repository** CRAN

**Date/Publication** 2019-12-17 23:00:05 UTC

## R topics documented:

---

## `delete_files_plink` *Delete all plink binary files*

---

### Description

This function deletes each of the plink binary files (BED/BIM/FAM extensions) given the shared base file path, warning with an informative message if any of the files did not exist.

### Usage

```
delete_files_plink(file)
```

### Arguments

| | |
|---|---|
| `file` | The shared file path (excluding BED/BIM/FAM extensions). |

### Value

Nothing

## Examples

```
# create dummy BED/BIM/FAM files
file <- tempfile('delete-me-test') # no extension
# add each extension and create empty files
file.create( paste0(file, '.bed') )
file.create( paste0(file, '.bim') )
file.create( paste0(file, '.fam') )

# delete the BED/BIM/FAM files we just created
delete_files_plink(file)
```

---

`genio` *genio (GENetics I/O): A package for reading and writing genetics data*

---

## Description

This package fully supports reading and writing plink BED/BIM/FAM files, as illustrated below. These functions make it easy to create dummy annotation tables to go with simulated genotype data too. Lastly, there is functionality to read and write Eigenstrat tables.

## Author(s)

**Maintainer**: Alejandro Ochoa <`alejandro.ochoa@duke.edu`> (ORCID)

## See Also

Useful links:

- `https://github.com/OchoaLab/genio`
- Report bugs at `https://github.com/OchoaLab/genio/issues`

## Examples

```
# read existing BED/BIM/FAM files

# first get path to BED file
file <- system.file("extdata", 'sample.bed', package = "genio", mustWork = TRUE)

# read genotypes and annotation tables
plink_data <- read_plink(file)
# genotypes
X <- plink_data$X
# locus annotations
bim <- plink_data$bim
# individual annotations
fam <- plink_data$fam

# the same works without .bed extension
```

```
file <- sub('\\.bed$', '', file) # remove extension
# it works!
plink_data <- read_plink(file)

# write data into new BED/BIM/FAM files
file_out <- tempfile('delete-me-example')
write_plink(file_out, X, bim, fam)

# delete example files when done
delete_files_plink(file_out)

# other functions not shown here allow reading and writing individual files,
# creating dummy tables to go with simulated genotypes,
# requiring the existence of these files,
# and reading and writing of Eigenstrat tables too.
```

---

## `ind_to_fam` *Convert an eigenstrat IND tibble into a plink FAM tibble*

---

### Description

This function takes an existing IND tibble and creates a FAM tibble with the same information and dummy values for missing data. In particular, the output FAM tibble will contain these columns with these contents:

**fam:** IND label

**id:** IND id

**pat:** `0` (missing paternal ID)

**mat:** `0` (missing maternal ID)

**sex:** IND sex converted to plink integer codes via `sex_to_int`

**peno:** `0` (missing phenotype)

As IND tibbles only contain the three columns listed above, there is no loss of information.

### Usage

```
ind_to_fam(ind)
```

### Arguments

| | |
|---|---|
| `ind` | The input eigenstrat IND tibble to convert. |

### Value

A plink FAM tibble.

## See Also

`sex_to_int`

Eigenstrat IND format reference: `https://github.com/DReichLab/EIG/tree/master/CONVERTF`

Plink FAM format reference: `https://www.cog-genomics.org/plink/1.9/formats#fam`

## Examples

```
# create a sample IND tibble
library(tibble)
ind <- tibble(
  id = 1:3,
  sex = c('U', 'M', 'F'),
  label = c(1, 1, 2)
)
# convert to FAM
fam <- ind_to_fam(ind)
# inspect:
fam
```

---

`make_bim` *Create a plink BIM tibble*

---

## Description

This function simplifies the creation of plink BIM-formatted tibbles, which autocompletes missing information if a partial tibble is provided, or generates a completely made up tibble if the number of individuals is provided. The default values are most useful for simulated genotypes, where IDs can be made up but must be unique, and there are no chromosomes, positions, or particular reference or alternative alleles.

## Usage

```
make_bim(tib, n = NA)
```

## Arguments

| | |
|---|---|
| `tib` | The input tibble (optional). Missing columns will be autocompleted with reasonable values that are accepted by plink and other external software. |
| `n` | The desired number of loci (rows). Required if `tib` is missing; otherwise it is ignored. |

## Details

Autocompleted column values:

**chr:** `1` (all data is on a single chromosome)

**id:** `1:n`

**posg:** `0` (missing)

**pos:** `1:n`

**ref:** `1`

**alt:** `2`

Note that $n$ is either given directly or obtained from the input tibble.

## Value

The input tibble with autocompleted columns and columns in default order, or the made up tibble if only the number of individuals was provided. The output begins with the standard columns in standard order: chr, id, posg, pos, ref, alt. Additional columns in the input tibble are preserved but placed after the standard columns.

## See Also

Plink BIM format reference: `https://www.cog-genomics.org/plink/1.9/formats#bim`

## Examples

```
# create a synthetic tibble for 10 loci
# (most common use case)
bim <- make_bim(n = 10)

# manually create a partial tibble with only chromosomes defined
library(tibble)
bim <- tibble(chr = 0:2)
# autocomplete the rest of the columns
bim <- make_bim(bim)
```

---

## `make_fam` *Create a plink FAM tibble*

---

## Description

This function simplifies the creation of plink FAM-formatted tibbles, which autocompletes missing information if a partial tibble is provided, or generates a completely made up tibble if the number of individuals is provided. The default values are most useful for simulated genotypes, where IDs can be made up but must be unique, and there are no parents, families, gender, or phenotype.

## Usage

```
make_fam(tib, n = NA)
```

## Arguments

| | |
|---|---|
| `tib` | The input tibble (optional). Missing columns will be autocompleted with reasonable values that are accepted by plink and other external software. |
| `n` | The desired number of individuals (rows). Required if `tib` is missing; otherwise it is ignored. |

## Details

Autocompleted column values:

**fam:** `1:n`

**id:** `1:n`

**pat:** `0` (missing)

**mat:** `0` (missing)

**sex:** `0` (missing)

**pheno:** `0` (missing)

Note that $n$ is either given directly or obtained from the input tibble.

## Value

The input tibble with autocompleted columns and columns in default order, or the made up tibble if only the number of individuals was provided. The output begins with the standard columns in standard order: fam, id, pat, mat, sex, pheno. Additional columns in the input tibble are preserved but placed after the standard columns.

## See Also

Plink FAM format reference: https://www.cog-genomics.org/plink/1.9/formats#fam

## Examples

```
# create a synthetic tibble for 10 individuals
# (most common use case)
fam <- make_fam(n = 10)

# manually create a partial tibble with only phenotypes defined
library(tibble)
fam <- tibble(pheno = 0:2)
# autocomplete the rest of the columns
fam <- make_fam(fam)
```

## read_bed *Read a genotype matrix in plink BED format*

### Description

This function reads genotypes encoded in a plink-formatted BED (binary) file, returning them in a standard R matrix containing genotypes (values in `c(0,1,2,NA)`). Each genotype per locus (m loci) and individual (n total) counts the number of alternative alleles or `NA` for missing data. No *.fam or *.bim files are read by this basic function. Since BED does not encode the data dimensions internally, these values must be provided by the user.

### Usage

```
read_bed(
  file,
  names_loci = NULL,
  names_ind = NULL,
  m_loci = NA,
  n_ind = NA,
  verbose = TRUE
)
```

### Arguments

| | |
|---|---|
| `file` | Input file path. *.bed extension may be omitted (will be added automatically if it is missing). |
| `names_loci` | Vector of loci names, to become the row names of the genotype matrix. If provided, its length sets `m_loci` below. If `NULL`, the returned genotype matrix will not have row names, and `m_loci` must be provided. |
| `names_ind` | Vector of individual names, to become the column names of the genotype matrix. If provided, its length sets `n_ind` below. If `NULL`, the returned genotype matrix will not have column names, and `n_ind` must be provided. |
| `m_loci` | Number of loci in the input genotype table. Required if `names_loci = NULL`, as its value is not inferrable from the BED file itself. Ignored if `names_loci` is provided. |
| `n_ind` | Number of individuals in the input genotype table. Required if `names_ind = NULL`, as its value is not inferrable from the BED file itself. Ignored if `names_ind` is provided. |
| `verbose` | If TRUE (default) function reports the path of the file being read (after autocompleting the extension). |

### Details

The code enforces several checks to validate data given the requested dimensions. Errors are thrown if file terminates too early or does not terminate after genotype matrix is filled. In addition, as each locus is encoded in an integer number of bytes, and each byte contains up to four individuals, bytes with fewer than four are padded with zeroes (non-zero pads throw errors).

This function only supports locus-major BED files, which are the standard for modern data. Format is validated via the BED file's magic numbers (first three bytes of file). Older BED files can be converted using plink.

## Value

The $m \times n$ genotype matrix.

## See Also

`read_plink` for reading a set of BED/BIM/FAM files.

Plink BED format reference: https://www.cog-genomics.org/plink/1.9/formats#bed

## Examples

```
# first obtain data dimensions from BIM and FAM files
# all file paths
file_bed <- system.file("extdata", 'sample.bed', package = "genio", mustWork = TRUE)
file_bim <- system.file("extdata", 'sample.bim', package = "genio", mustWork = TRUE)
file_fam <- system.file("extdata", 'sample.fam', package = "genio", mustWork = TRUE)
# read annotation tables
bim <- read_bim(file_bim)
fam <- read_fam(file_fam)

# read an existing plink *.bim file
# pass locus and individual IDs as vectors, setting data dimensions too
X <- read_bed(file_bed, bim$id, fam$id)
X

# can specify without extension
file_bed <- sub('\\.bed$', '', file_bed) # remove extension from this path on purpose
file_bed # verify .bed is missing
X <- read_bed(file_bed, bim$id, fam$id) # loads too!
X
```

---

`read_bim` *Read plink *.bim files*

---

## Description

This function reads a standard plink *.bim file into a tibble. It uses readr::read_table2 to do it efficiently.

## Usage

```
read_bim(file, verbose = TRUE)
```

## Arguments

| | |
|---|---|
| `file` | Input file (whatever is accepted by readr::read_table2). If file as given does not exist and is missing the expected *.bim extension, the function adds the .bim extension and uses that path if that file exists. Additionally, the .gz extension is added automatically if the file (after *.bim extension is added as needed) is still not found and did not already contained the .gz extension and adding it points to an existing file. |
| `verbose` | If TRUE (default) function reports the path of the file being loaded (after autocompleting the extensions). |

## Value

A tibble with columns: chr, id, posg, pos, ref, alt

## See Also

`read_plink` for reading a set of BED/BIM/FAM files.

Plink BIM format reference: `https://www.cog-genomics.org/plink/1.9/formats#bim`

## Examples

```
# read an existing plink *.bim file
file <- system.file("extdata", 'sample.bim', package = "genio", mustWork = TRUE)
bim <- read_bim(file)
bim

# can specify without extension
file <- sub('\\.bim$', '', file) # remove extension from this path on purpose
file # verify .bim is missing
bim <- read_bim(file) # loads too!
bim
```

---

## `read_fam` *Read plink *.fam files*

---

## Description

This function reads a standard plink *.fam file into a tibble. It uses readr::read_table2 to do it efficiently.

## Usage

```
read_fam(file, verbose = TRUE)
```

### Arguments

| | |
|---|---|
| `file` | Input file (whatever is accepted by readr::read_table2). If file as given does not exist and is missing the expected *.fam extension, the function adds the .fam extension and uses that path if that file exists. Additionally, the .gz extension is added automatically if the file (after *.fam extension is added as needed) is still not found and did not already contained the .gz extension and adding it points to an existing file. |
| `verbose` | If TRUE (default) function reports the path of the file being loaded (after autocompleting the extensions). |

### Value

A tibble with columns: fam, id, pat, mat, sex, pheno.

### See Also

`read_plink` for reading a set of BED/BIM/FAM files.

Plink FAM format reference: `https://www.cog-genomics.org/plink/1.9/formats#fam`

### Examples

```
# read an existing plink *.fam file
file <- system.file("extdata", 'sample.fam', package = "genio", mustWork = TRUE)
fam <- read_fam(file)
fam

# can specify without extension
file <- sub('\\.fam$', '', file) # remove extension from this path on purpose
file # verify .fam is missing
fam <- read_fam(file) # load it anyway!
fam
```

## `read_ind` *Read eigenstrat *.ind files*

### Description

This function reads a standard eigenstrat *.ind file into a tibble. It uses readr::read_table2 to do it efficiently.

### Usage

```
read_ind(file, verbose = TRUE)
```

## Arguments

| | |
|---|---|
| `file` | Input file (whatever is accepted by readr::read_table2). If file as given does not exist and is missing the expected *.ind extension, the function adds the .ind extension and uses that path if that file exists. Additionally, the .gz extension is added automatically if the file (after *.ind extension is added as needed) is still not found and did not already contained the .gz extension and adding it points to an existing file. |
| `verbose` | If TRUE (default) function reports the path of the file being loaded (after autocompleting the extensions). |

## Value

A tibble with columns: id, sex, label.

## See Also

Eigenstrat IND format reference: https://github.com/DReichLab/EIG/tree/master/CONVERTF

## Examples

```
# read an existing eigenstrat *.ind file
file <- system.file("extdata", 'sample.ind', package = "genio", mustWork = TRUE)
ind <- read_ind(file)
ind

# can specify without extension
file <- sub('\\.ind$', '', file) # remove extension from this path on purpose
file # verify .ind is missing
ind <- read_ind(file) # load it anyway!
ind
```

---

## `read_phen` *Read *.phen files*

---

## Description

This function reads a standard *.phen file into a tibble. It uses readr::read_table2 to do it efficiently. GCTA and EMMAX use this format.

## Usage

```
read_phen(file, verbose = TRUE)
```

## Arguments

| | |
|---|---|
| `file` | Input file (whatever is accepted by readr::read_table2). If file as given does not exist and is missing the expected *.phen extension, the function adds the .phen extension and uses that path if that file exists. Additionally, the .gz extension is added automatically if the file (after *.phen extension is added as needed) is still not found and did not already contained the .gz extension and adding it points to an existing file. |
| `verbose` | If TRUE (default) function reports the path of the file being loaded (after auto-completing the extensions). |

## Value

A tibble with columns: fam, id, pheno.

## See Also

GCTA PHEN format reference: `https://cnsgenomics.com/software/gcta/#GREMLanalysis`

## Examples

```
# read an existing plink *.phen file
file <- system.file("extdata", 'sample.phen', package = "genio", mustWork = TRUE)
phen <- read_phen(file)
phen

# can specify without extension
file <- sub('\\.phen$', '', file) # remove extension from this path on purpose
file # verify .phen is missing
phen <- read_phen(file) # load it anyway!
phen
```

---

`read_plink` *Read genotype and sample data in a plink BED/BIM/FAM file set.*

---

## Description

This function reads a genotype matrix (X) and its associated locus (bim) and individual (fam) data tables in the three plink files in BED, BIM, and FAM formats, respectively. All inputs must exist or an error is thrown. This function is a wrapper around the more basic functions `read_bed`, `read_bim`, `read_fam`. Below suppose there are $m$ loci and $n$ individuals.

## Usage

```
read_plink(file, verbose = TRUE)
```

## Arguments

| | |
|---|---|
| `file` | Input file path, without extensions (each of .bed, .bim, .fam extensions will be added automatically as needed). Alternatively, input file path may have .bed extension (but not .bim, .fam, or other extensions). |
| `verbose` | If TRUE (default) function reports the paths of the files being read (after autocompleting the extensions). |

## Value

A named list with items in this order: X (genotype matrix), bim (tibble), fam (tibble). X has row and column names corresponding to the `id` values of the bim and fam tibbles.

## See Also

`read_bed`, `read_bim`, `read_fam`.

Plink BED/BIM/FAM format reference: `https://www.cog-genomics.org/plink/1.9/formats`

## Examples

```
# first get path to BED file
file <- system.file("extdata", 'sample.bed', package = "genio", mustWork = TRUE)

# read genotypes and annotation tables
plink_data <- read_plink(file)
# genotypes
plink_data$X
# locus annotations
plink_data$bim
# individual annotations
plink_data$fam

# the same works without .bed extension
file <- sub('\\.bed$', '', file) # remove extension
# it works!
plink_data <- read_plink(file)
```

---

## `read_snp` *Read eigenstrat *.snp files*

---

## Description

This function reads a standard eigenstrat *.snp file into a tibble. It uses readr::read_table2 to do it efficiently.

## Usage

```
read_snp(file, verbose = TRUE)
```

### Arguments

| | |
|---|---|
| `file` | Input file (whatever is accepted by readr::read_table2). If file as given does not exist and is missing the expected *.snp extension, the function adds the .snp extension and uses that path if that file exists. Additionally, the .gz extension is added automatically if the file (after *.snp extension is added as needed) is still not found and did not already contained the .gz extension and adding it points to an existing file. |
| `verbose` | If TRUE (default) function reports the path of the file being loaded (after autocompleting the extensions). |

### Value

A tibble with columns: id, chr, posg, pos, ref, alt

### See Also

Eigenstrat SNP format reference: `https://github.com/DReichLab/EIG/tree/master/CONVERTF`

### Examples

```
# read an existing eigenstrat *.snp file
file <- system.file("extdata", 'sample.snp', package = "genio", mustWork = TRUE)
snp <- read_snp(file)
snp

# can specify without extension
file <- sub('\\.snp$', '', file) # remove extension from this path on purpose
file # verify .snp is missing
snp <- read_snp(file) # load it anyway!
snp
```

## `require_files_plink` *Require that plink binary files are present*

### Description

This function checks that each of the plink binary files (BED/BIM/FAM extensions) are present, given the shared base file path, stopping with an informative message if any of the files is missing. This function aids troubleshooting, as various downstream external software report missing files differently and sometimes using confusing or obscure messages.

### Usage

```
require_files_plink(file)
```

### Arguments

| | |
|---|---|
| `file` | The shared file path (excluding BED/BIM/FAM extensions). |

### Value

Nothing

### Examples

```
# check that the samples we want exist
# start with bed file
file <- system.file("extdata", 'sample.bed', package = "genio", mustWork = TRUE)
# remove extension
file <- sub('\\.bed$', '', file)
# since all sample.{bed,bim,fam} files exist, this will not stop with error messages:
require_files_plink(file)
```

---

## `sex_to_char` *Convert integer sex codes to character codes*

---

### Description

This function accepts the integer sex codes accepted by plink and turns them into the character codes accepted by eigenstrat. Only upper-case characters are returned. The correspondence is:

**0:** U (unknown)

**1:** M (male)

**2:** F (female)

Any other cases will also be mapped to U (unknown) but with a warning (0 does not generate warnings).

### Usage

```
sex_to_char(sex)
```

### Arguments

| | |
|---|---|
| `sex` | Integer vector of sex codes |

### Value

The converged character vector of sex codes

### See Also

`sex_to_int`

Eigenstrat IND format reference: https://github.com/DReichLab/EIG/tree/master/CONVERTF

Plink FAM format reference: https://www.cog-genomics.org/plink/1.9/formats#fam

## Examples

```
# verify the mapping above
sex_int <- 0:2
sex_char <- c('U', 'M', 'F') # expected values
stopifnot(
  all(
    sex_to_char( sex_int ) == sex_char
  )
)
```

---

## sex_to_int *Convert character sex codes to integer codes*

---

### Description

This function accepts the character sex codes accepted by eigenstrat and turns them into the integer codes accepted by plink. Matching is case insensitive. The correspondence is:

**U:** 0 (unknown)

**M:** 1 (male)

**F:** 2 (female)

Any other characters will also be mapped to 0 (unknown) but with a warning (U does not generate warnings).

### Usage

```
sex_to_int(sex)
```

### Arguments

|  |  |
|---|---|
| `sex` | Character vector of sex codes |

### Value

The converged numeric vector of sex codes

### See Also

`sex_to_char`

Eigenstrat IND format reference: `https://github.com/DReichLab/EIG/tree/master/CONVERTF`

Plink FAM format reference: `https://www.cog-genomics.org/plink/1.9/formats#fam`

## Examples

```
# verify the mapping above
sex_char <- c('U', 'm', 'f') # mixed case works!
sex_int <- 0:2 # expected values
stopifnot(
  all(
    sex_to_int( sex_char ) == sex_int
  )
)
```

---

## write_bed *Write a genotype matrix into plink BED format*

---

### Description

This function accepts a standard R matrix containing genotypes (values in `c(0,1,2,NA)`) and writes it into a plink-formatted BED (binary) file. Each genotype per locus (m loci) and individual (n total) counts the number of alternative alleles or `NA` for missing data. No *.fam or *.bim files are created by this basic function.

### Usage

```
write_bed(file, X, verbose = TRUE)
```

### Arguments

| | |
|---|---|
| `file` | Output file path. .bed extension may be omitted (will be added automatically if it is missing). |
| `X` | The $m \times n$ genotype matrix. Row and column names, if present, are ignored. |
| `verbose` | If TRUE (default) function reports the path of the file being written (after auto-completing the extension). |

### Details

Genotypes with values outside of $[0, 2]$ cause an error, in which case the partial output is deleted. However, beware that decimals get truncated internally, so values that truncate to 0, 1, or 2 will not raise errors. The BED format does not accept fractional dosages, so such data will not be written as expected.

### Value

Nothing

### See Also

`write_plink` for writing a set of BED/BIM/FAM files.

Plink BED format reference: `https://www.cog-genomics.org/plink/1.9/formats#bed`

## Examples

```
file_out <- tempfile('delete-me-example', fileext = '.bed') # will also work without extension
# create 10 random genotypes
X <- rbinom(10, 2, 0.5)
# replace 3 random genotypes with missing values
X[sample(10, 3)] <- NA
# turn into 5x2 matrix
X <- matrix(X, nrow = 5, ncol = 2)
# write this data to file in BED format
# (only *.bed gets created, no *.fam or *.bim in this call)
write_bed(file_out, X)
# delete output when done
file.remove(file_out)
```

---

## `write_bim` *Write plink *.bim files*

---

### Description

This function writes a tibble with the right columns into a standard plink *.bim file. It uses readr::write_tsv to do it efficiently.

### Usage

```
write_bim(file, tib, verbose = TRUE)
```

### Arguments

| | |
|---|---|
| `file` | Output file (whatever is accepted by readr::write_tsv). If file is missing the expected *.bim extension, the function adds it. |
| `tib` | The tibble or data.frame to write. It must contain these columns: chr, id, posg, pos, ref, alt. Throws an error if any of these columns are missing. Additional columns are ignored. Columns are automatically reordered in output as expected in format. |
| `verbose` | If TRUE (default) function reports the path of the file being written (after autocompleting the extension). |

### Value

The output `tib` invisibly (what readr::write_tsv returns).

### See Also

`write_plink` for writing a set of BED/BIM/FAM files.

Plink BIM format reference: `https://www.cog-genomics.org/plink/1.9/formats#bim`

## Examples

```
# create a dummy tibble with the right columns
library(tibble)
tib <- tibble(
    chr = 1:3,
    id = 1:3,
    posg = 0,
    pos = 1:3,
    ref = 'A',
    alt = 'B'
)
# a dummy file
file_out <- tempfile('delete-me-example', fileext = '.bim') # will also work without extension
# write the table out in *.bim format (no header, columns in right order)
write_bim(file_out, tib)
# delete output when done
file.remove(file_out)
```

---

`write_fam` *Write plink *.fam files*

---

## Description

This function writes a tibble with the right columns into a standard plink *.fam file. It uses readr::write_tsv to do it efficiently.

## Usage

```
write_fam(file, tib, verbose = TRUE)
```

## Arguments

| | |
|---|---|
| `file` | Output file (whatever is accepted by readr::write_tsv). If file is missing the expected *.fam extension, the function adds it. |
| `tib` | The tibble or data.frame to write. It must contain these columns: fam, id, pat, mat, sex, pheno. Throws an error if any of these columns are missing. Additional columns are ignored. Columns are automatically reordered in output as expected in format. |
| `verbose` | If TRUE (default) function reports the path of the file being written (after autocompleting the extension). |

## Value

The output `tib` invisibly (what readr::write_tsv returns).

### See Also

write_plink for writing a set of BED/BIM/FAM files.

Plink FAM format reference: https://www.cog-genomics.org/plink/1.9/formats#fam

### Examples

```
# create a dummy tibble with the right columns
library(tibble)
tib <- tibble(
    fam = 1:3,
    id = 1:3,
    pat = 0,
    mat = 0,
    sex = 1,
    pheno = 1
)
# a dummy file
file_out <- tempfile('delete-me-example', fileext = '.fam') # will also work without extension
# write the table out in *.fam format (no header, columns in right order)
write_fam(file_out, tib)
# delete output when done
file.remove(file_out)
```

## write_ind *Write eigenstrat *.ind files*

### Description

This function writes a tibble with the right columns into a standard eigenstrat *.ind file. It uses readr::write_tsv to do it efficiently.

### Usage

```
write_ind(file, tib, verbose = TRUE)
```

### Arguments

| | |
|---|---|
| file | Output file (whatever is accepted by readr::write_tsv). If file is missing the expected *.ind extension, the function adds it. |
| tib | The tibble or data.frame to write. It must contain these columns: id, sex, label. Throws an error if any of these columns are missing. Additional columns are ignored. Columns are automatically reordered in output as expected in format. |
| verbose | If TRUE (default) function reports the path of the file being written (after autocompleting the extension). |

## Value

The output `tib` invisibly (what readr::write_tsv returns).

## See Also

Eigenstrat IND format reference: https://github.com/DReichLab/EIG/tree/master/CONVERTF

## Examples

```
# create a dummy tibble with the right columns
library(tibble)
tib <- tibble(
    id = 1:3,
    sex = 1,
    label = 1
)
# a dummy file
file_out <- tempfile('delete-me-example', fileext = '.ind') # will also work without extension
# write the table out in *.ind format (no header, columns in right order)
write_ind(file_out, tib)
# delete output when done
file.remove(file_out)
```

---

`write_phen` *Write *.phen files*

---

## Description

This function writes a tibble with the right columns into a standard *.phen file. It uses readr::write_tsv to do it efficiently. GCTA and EMMAX use this format.

## Usage

```
write_phen(file, tib, verbose = TRUE)
```

## Arguments

| | |
|---|---|
| `file` | Output file (whatever is accepted by readr::write_tsv). If file is missing the expected *.phen extension, the function adds it. |
| `tib` | The tibble or data.frame to write. It must contain these columns: fam, id, pheno. Throws an error if any of these columns are missing. Additional columns are ignored. Columns are automatically reordered in output as expected in format. |
| `verbose` | If TRUE (default) function reports the path of the file being written (after autocompleting the extension). |

## Value

The output `tib` invisibly (what readr::write_tsv returns).

### See Also

GCTA PHEN format reference: https://cnsgenomics.com/software/gcta/#GREMLanalysis

### Examples

```
# create a dummy tibble with the right columns
library(tibble)
tib <- tibble(
    fam = 1:3,
    id = 1:3,
    pheno = 1
)
# a dummy file
file_out <- tempfile('delete-me-example', fileext = '.phen') # will also work without extension
# write the table out in *.phen format (no header, columns in right order)
write_phen(file_out, tib)
# delete output when done
file.remove(file_out)
```

---

## write_plink *Write genotype and sample data into a plink BED/BIM/FAM file set.*

---

### Description

This function writes a genotype matrix (X) and its associated locus (bim) and individual (fam) data tables into three plink files in BED, BIM, and FAM formats, respectively. This function is a wrapper around the more basic functions `write_bed`, `write_bim`, `write_fam`, but additionally tests that the data dimensions agree (or stops with an error). Also checks that the genotype row and column names agree with the bim and fam tables if they are all present. In addition, if `bim = NULL` or `fam = NULL`, these are auto-generated using `make_bim` and `make_fam`, which is useful behavior for simulated data. Lastly, the phenotype can be provided as a separate argument and incorporated automatically if `fam=NULL` (a common scenario for simulated genotypes and traits). Below suppose there are $m$ loci and $n$ individuals.

### Usage

```
write_plink(file, X, bim = NULL, fam = NULL, pheno = NULL, verbose = TRUE)
```

### Arguments

| | |
|---|---|
| `file` | Output file path, without extensions (each of .bed, .bim, .fam extensions will be added automatically as needed). |
| `X` | The $m \times n$ genotype matrix. |
| `bim` | The tibble or data.frame containing locus information. It must contain $m$ rows and these columns: chr, id, posg, pos, ref, alt. If NULL (default), it will be quietly auto-generated. |

| | |
|---|---|
| `fam` | The tibble or data.frame containing individual information. It must contain $n$ rows and these columns: fam, id, pat, mat, sex, pheno. If NULL (default), it will be quietly auto-generated. |
| `pheno` | The phenotype to write into the FAM file assuming `fam=NULL`. This must be a length-$n$ vector. This will be ignored (with a warning) if `fam` is provided. |
| `verbose` | If TRUE (default) function reports the paths of the files being written (after autocompleting the extensions). |

## Value

Invisibly, a named list with items in this order: X (genotype matrix), bim (tibble), fam (tibble). This is most useful when either BIM or FAM tables were auto-generated.

## See Also

`write_bed`, `write_bim`, `write_fam`, `make_bim`, `make_fam`.

Plink BED/BIM/FAM format reference: `https://www.cog-genomics.org/plink/1.9/formats`

## Examples

```
# here is an example for a simulation

# create 10 random genotypes
X <- rbinom(10, 2, 0.5)
# replace 3 random genotypes with missing values
X[sample(10, 3)] <- NA
# turn into 5x2 matrix
X <- matrix(X, nrow = 5, ncol = 2)

# simulate a trait for two individuals
pheno <- rnorm(2)

# write this data to BED/BIM/FAM files
# output path without extension
file_out <- tempfile('delete-me-example')
# here all of the BIM and FAM columns except `pheno` are autogenerated
write_plink(file_out, X, pheno = pheno)

# delete all three outputs when done
delete_files_plink( file_out )
```

---

# `write_snp` *Write eigenstrat *.snp files*

---

## Description

This function writes a tibble with the right columns into a standard eigenstrat *.snp file. It uses readr::write_tsv to do it efficiently.

## Usage

```
write_snp(file, tib, verbose = TRUE)
```

## Arguments

| | |
|---|---|
| `file` | Output file (whatever is accepted by readr::write_tsv). If file is missing the expected *.snp extension, the function adds it. |
| `tib` | The tibble or data.frame to write. It must contain these columns: id, chr, posg, pos, ref, alt Throws an error if any of these columns are missing. Additional columns are ignored. Columns are automatically reordered in output as expected in format. |
| `verbose` | If TRUE (default) function reports the path of the file being written (after autocompleting the extension). |

## Value

The output `tib` invisibly (what readr::write_tsv returns).

## See Also

Eigenstrat SNP format reference: https://github.com/DReichLab/EIG/tree/master/CONVERTF

## Examples

```
# create a dummy tibble with the right columns
library(tibble)
tib <- tibble(
    id = 1:3,
    chr = 1:3,
    posg = 0,
    pos = 1:3,
    ref = 'A',
    alt = 'B'
)
# a dummy file
file_out <- tempfile('delete-me-example', fileext = '.snp') # will also work without extension
# write the table out in *.snp format (no header, columns in right order)
write_snp(file_out, tib)
# delete output when done
file.remove(file_out)
```

# Index