PLATO User Guide

Current version: PLATO 2.0

Last modified: August 2014

Ritchie Lab, Center for Systems Genomics

Pennsylvania State University, University Park, PA 16802

URL: https://ritchielab.psu.edu/

Email: software@ritchielab.psu.edu

Table of Contents

Overview

PLATO (the PLatform for the Analysis, Translation, and Organization of large-scale data) is analysis software designed to efficiently and easily analyze genome-wide genetic data. PLATO is designed to be both modular and extensible, making it easy to adapt to the ever-changing analysis requests of bioinformaticians.

The core functionality of PLATO is encapsulated in a series of commands, specified by the user. Each command is processed serially, either analyzing, loading, or printing genotypic data. By stringing together a series of commands, the user can customize the analysis pipeline to suit their particular needs. The convention for calling PLATO on the command line is:

```
$ plato [General Options] \
        command1 [command1 options] \
        command2 [command2 options] ...
```

This manual will document all of the available commands as well as all of the options available. To get started, we recommend looking at the examples below as well as the load-data or load-trait commands, which load genetic data and numeric data into PLATO for analysis. On the command line, options are prefixed with either one or two dashes (- or --) for the short or long form of the option. All options will have a long form, but not all options will have a short form. Commands given to PLATO are not prefixed at all, and any options seen after the command will be passed to only that command.

When documenting the options for each command, we will use the following convention:

--<option> [-<short name>] <argument type> (=<default value>)

Here, the option is the option name, with the short name given if it exists. If the option takes an argument, the type of the argument will be provided, with the type being one of the following:

- **string**
  A character string value. Any restrictions on the string (filename, etc.) will be specified in the description of the option. If the string includes whitespace, it should be enclosed with double quotes ("") or it will not be parsed correctly.
- **int**
  An integer value. In most cases, the integer must be non-negative; if negative integers are allowed for this parameter, it will be explicitly stated in the option description.
- **float**
  A floating-point (decimal) number. In most cases, only non-negative entries make sense, but PLATO does not enforce this restriction.
- **enum**
  A string value with a very limited set of valid inputs. When this argument is seen, the description of the option will describe all valid inputs for this parameter.

When a default value is given, this is the value that will be used if the option is not provided on the command line. The default values are useful for necessary options such as thresholds and output file names which need values, but the user may not want to provide all of the options for compelete customization.

Examples

Below are some examples of analyses that can be run in PLATO and the options that are available to the user. These examples are intended as a jumping off point for the user to customize their own analyses. For a handy list of all available commands and options, organized by function, see the PLATO Quick Reference, which lists the commands by function and is cross-referenced to their location in the manual.

Recode PED/MAP to Binary PLINK format

```
$ plato load-data --file geno output-bed --file binarygeno
```

Recode Binary PLINK to PED/MAP, Only a Single Chromosome

```
$ plato load-data --bfile bingeno --chrom 22 \
        output-ped --ped geno.ped --map custom_geno.map
```

Perform a Simple Case/Control GWAS (no covariates, Bonferroni corrected p-values)

```
$ plato load-data --bfile bingeno \
        logistic --correction Bonferroni
```

PheWAS on Quantitative Outcomes using Markers with MAF between 10% and 40%

```
$ plato load-data --bfile bingeno \
        load-trait --file pheno.txt --missing="-999" \
        filter-maf --min 0.1 --max 0.4
        linear --phewas --covariates AGE,BMI
```

EWAS (No Genetic Data, Quantitative Predictor Variables)

```
$ plato load-trait --file pheno.txt --dummy-samples --extra-samples
        logistic --exclude-markers --use-traits --outcome PHENO
```

Concordance Checking (VCF vs. PED/MAP w/o FID)

```
$ plato load-data --vcf-file data.vcf.gz
        concordance --file geno --no-fid
```

GWAS with Dog Data and Permutation Testing

```
$ plato --chroms 38 load-data --file dog_geno
        logistic --permutations 1000
```

PLATO Quick Reference

General Options:

Special Options	
-h [--help]	Display help message
-L [--list-command]	List all available PLATO commands
-C [--help-command] cmd	Display help for the given command
-v [--version]	Display version
Global Options	
-f [--logfile] arg (=plato.log)	Name of the log file
--chroms (=22)	Number of autosomal chromosomes
--extra-chroms (=X,Y,XY,M-MT)	Comma-separated list of extra chromosomes; dashes indicate aliases

Available Commands:

Data Loading	
load-data	Loads genetic data into PLATO
load-trait	Loads numeric data into PLATO
Data Transformation	
recode-alleles	Set the referent allele for markers
Data Output	
output-beagle	Output data in BEAGLE format
output-bed	Output data in binary PLINK (bed/bim/fam) format
output-eigenstrat	Output data in Eigenstrat format
output-ped	Output data in PLINK (ped/map) format
output-tped	Output data in transposed PLINK (tped/tfam) format
Filtering	
filter-maf	Filter markers based on their minor allele frequency
filter-marker-call	Filter markers based on their call rate (missing rate)
filter-sample-call	Filter samples based on their call rate (missing rate)
filter-trait-missing	Filter traits based on the missing rate
Analysis	
batch	Execute PLATO commands from a file
concordance	Check concordance between two datasets
linear	Perform linear regression on the data
logistic	Perform logistic regression on the data
regress-auto	Perform linear or logistic regression, based on the phenotype

Installation

PLATO is packaged with the GNU autotools, so installation occurs in four steps: unpacking, configuration, compilation and installation. Each of those steps will be described below, but first the user must ensure that the prerequisites for running PLATO are met.

Prerequisites

The following are prerequisites for building and running PLATO.

- A modern C++ compiler.
- Boost Libraries for C++, version 1.42 or later (http://www.boost.org).
- Gnu Scientific Library (http://www.gnu.org/software/gsl/).
- (optional) An MPI-enabled compiler and library suite.

Unpacking

PLATO is distributed as a zipped tarball, and the command for unpacking the distribution is:

```
$ tar -xvzf plato-2.0.0.tar.gz
```

This will unpack the source code into a directory called `plato-2.0.0`. For all of the following commands, we assume that you are in this directory.

```
$ cd ./plato-2.0.0
```

Configuration

In order to compile PLATO, the user must first configure the software. This script will attempt to detect all of the prerequisites on the user's system, and this is the time for the user to specify system-specific options, such as the location of the installed program. The command is:

```
$ ./configure
```

The configure script can also take a number of helpful options, some of which are detailed below:

--help
This option will list all of the available options that can be passed to the configure script.

--prefix=[path]
This option tells PLATO to install itself into the given path, which is useful if you do not have administrative access to the computer. By default, the program will be in [path]/bin. Note: when using this option, the path given must be an absolute path and cannot use any shell expansions, such as the "~" notation.

--enable-debug
For the advanced users, this option will turn off all optimization and turn on debugging symbols, which can be helpful in diagnosing a problem with the PLATO software.

Compilation and Installation

Once the configuration of PLATO is complete, the user can then compile and install the software. To compile PLATO, simply use the "make" command. Note that to speed compilation, a user may use the "-j N" option to make in order to specify a maximum of N simultaneous parallel processes:

```
$ make
```

Once compilation is complete, PLATO will be available in the current directory. Typically, the user will need administrative rights to complete the installation step. In order to move PLATO to the directory specified by the configuration step, type:

```
# make install
```

Running PLATO

After PLATO is installed, the user should be able to run PLATO from the command line from any directory. If PLATO is not found, the installation directory may need to be added to the user's PATH environment variable.

This section describes the special and global options available to the user. These options should be given before any commands on the command line; if any special commands are given, no commands are necessary.

Parallel Options

In PLATO, certain commands may use parallelization in order to speed computation times. PLATO utilizes a master/worker parallelization paradigm, where a single computational thread parcels out chunks of data to multiple worker processes, who perform the computation and return the result to the master thread, which compiles the overall result.

There are two types of parallelization that PLATO commands may implement: threaded (shared memory) and interprocess (distributed memory). Threaded commands will typically have the "--threads" option available on the command line, and these will require multiple cores on the same machine. This option requires less overhead, and can substantially speed up computation, but it does not scale beyond the number of processors on a single machine. To scale beyond a single machine, PLATO can use interprocess parallelization using Message Passing Interface (MPI). Those commands that are capable of MPI parallelization will be specifically noted in the manual; all other commands will run in serial when using MPI. To use MPI, the user must have an MPI compiler installed on their machine during the configuration step; PLATO will automatically attempt to enable MPI when it is available. Additionally, the user must run plato through "mpirun" as follows:

```
$ mpirun -n N plato ...
```

Above, "N" is the number of processes used by PLATO, and the ellipsis indicates that all of the remaining options to PLATO remain the same. Note that when using MPI, N should be greater than 2, as one processor is solely devoted to dividing the work appropriately. Additionally, by combining "--threads" and "mpirun", PLATO can use multiple threads on each processor running the work, further speeding processing time.

Special Options

The following options are special options that allow PLATO to print help messages. If any of the following commands are given, no commands provided will be run. These arguments are listed below in their order of priority.

--help [-h]

Prints some basic help messages. This is a good place to start for new users getting used to PLATO.

--list-commands [-L]

Lists all commands available in the current version of PLATO. In the case of a discrepancy between the manual, the help documentation in the PLATO program should be considered correct.

--help-command [-C] string

Prints the help documentation for a given command. The help documentation will list ALL available options for a given command. While every effort is taken to synchronize the manual to the available PLATO options, there may be experimental or undocumented options available that will be shown using this option. Note that if no command is given to this option, PLATO will print out the help documentation for all available commands.

--version [-v]

Prints a brief version string, along with a copyright message and a bug-reporting email.

Global Options

The following options affect global settings for how data is stored and the location of files used by all commands. Use of the options will not prevent commands from processing, but may affect how data is represented and presented to the user.

--logfile [-f] string (=plato.log)

Gives the name of the log file for this run of PLATO. All informational messages, warnings, and errors printed to the screen will also be printed to the log file so that a user can have a record of each PLATO run.

--chroms int (=22)

Gives the number of autosomes in the organism for which you are loading genotypic data. By default, PLATO was designed for human data and uses 22 autosomes.

--extra-chroms string (=X,Y,XY,M-MT)

Provides a comma-separated string of non-autosomal chromosomes to use in the analysis. You may use dashes to indicate aliases for a chromosome, and chromosmes are case-insensitive. For each extra chromosome given, PLATO will map it to the next number after the number of autosomes. In the default configuration, chromosome X is chromosome 23, Y is mapped to chromosome 24, chromosome XY (the pseudo-autosomal region of the sex chromosomes) is mapped to chromosome 25, and chromosome M or MT (mitochondrial chromosome) is chromosome number 26.

Commands

In order to do useful work with PLATO, a user must enter one or more commands. Commands are entered on the command line by beginning an argument without an argument escape (- or --). Any arguments beginning with the argument escape will be processed according to the most recent command. Because some command sequences can get long, commands as entered on the command line will be printed in the log file, one per line. This section will describe all of the available commands in PLATO along with the available options for each command.

batch

The batch command is a way for the user to save a sequence of commands to a file to be reused. For those familiar with PLATO 1.x, this file is similar to the required batch file.

--file [-f] string

This option gives the batch command the name of the file containing commands to process. Each batch file must contain a sequence of commands, one per line. Below is an example of a batch file that performs some basic QC filtering and then runs a logistic regression:

```
filter-marker-call --threshold 0.99
filter-maf --min 0.05 --max 0.4
logisitc --output myResults.txt
```

concordance

The concordance command will check concordance between the currently loaded dataset and a new dataset, to be loaded with this command. For the data input options, see the load-data command. There are three distinct types of discordance possible:

- **Sample mismatch**
  A sample is in one dataset but not another

- **Marker mismatch**
  A marker is present in one dataset but not another

- **Call discordance**
  The called genotype for a particular sample/marker combination does not match across the two datasets.

This command will produce a single output for the first two kinds of mismatch, and three outputs for the actual call discordance. The three discordance outputs are a file of every discordant call and a summary of discordance by sample and by marker. Any of the file output can be suppressed by passing an empty string (“”) to the appropriate argument. The naming convention for the files is <prefix>.<suffix>.<extension>, all of which are customizable below.

--prefix string (=concordance)

This option is the prefix that all concordance files will begin with. Note that this argument may include directories if the user wishes to place concordance files in a separate directory.

--error string (=error)

This argument gives the suffix for the file of every discordant call.

--sample string (=sample)

Gives the suffix for the discordance summary by sample.

--marker string (=marker)

Gives the suffix for the discordance summary by marker.

--marker-mismatch string (=marker-mismatch)

Gives the suffix for the marker mismatch file.

--sample-mismatch string (=sample-mismatch)

Gives the suffix for the sample mismatch file.

--extension string (=txt)

Gives the extension for all concordance output files.

--inc-missing

If given, the concordance command will consider calls that are present in one dataset and missing in the second as discordant. By default, the concordance checking is performed only on calls that are present in both datasets.

--sep string (=<TAB>)

Gives the string to use for separating columns in concordance files. Useful for automated processing.

filter-maf

The filter-maf command will set markers as disabled if they do not meet the minor allele frequency thresholds set by the user. All markers marked as disabled will be excluded from further analysis, including output. **Important Note:** This command assumes a biallelic marker, and the minor allele frequency is the minimum of the referent allele frequency and one minus the referent allele frequency.

--min float (=0.05)

Drop all markers with a minor allele frequency below this value (default is a MAF of 5%). Set to 0 to disable dropping markers based on low minor allele frequency.

--max float (=1.0)

Drop all markers with a minor allele frequency above this value (default is to keep all markers). This can be useful in isolating uncommon or rare variants in the dataset.

filter-marker-call

This command sets makers as disabled according to their missing rate. Note that even though a marker has been marked as disabled, it can still be explicitly included in models tested by either linear or logistic through the use of model files.

--threshold float (=0.99)

Drops all markers with a call rate below the given threshold (default is 99%, or a 1% missing rate).

filter-sample-call

This command sets samples as disabled according to their missing rate. Note that if the data given is pair or trio data, this can adversely affect certain downstream analyses, such as output-beagle.

--threshold float (=0.99)

Drops all samples with a call rate below the given threshold (default is 99%, or a 1% missing rate).

filter-trait-missing

This command drops all traits according to their missing rate. Note that if a trait is disabled through the use of this command, it is still possible to include it as a covariate in further analyses or to use it explicitly as a variable though the use of a model file in the regression commands.

--threshold float (=0.99)

Drops all traits with a call rate below the given threshold (default is 99%, or a 1% missing rate).

linear

This command performs a simple (ordinary least-squares) regression on the data previously loaded into PLATO. Most of the options outlined here also apply to the logistic command as well; any options that are specific to this command will be noted.

The strength of PLATO as compared with other available tools is the ability to automatically generate models to perform statistical hypothesis testing. By default, PLATO will perform a standard GWAS, but it can be configured to perform an EWAS, GxG, GxE, or PheWAS analysis depending on the options given. See the Model Generation for Regression section for details on how to properly specify the desired analysis type. Additionally, PLATO can perform permutation testing; for details on how to enable this feature, see the Permutation Testing section later in the manual.

NOTE: This command is MPI-enabled.

--interactions

If given, include the interaction terms between the variables of interest in the models. Note that this argument has no effect if the generated models have only one variable.

--covariates string

Provides a comma-separated list of traits to include as covariates in each model. Covariates given in this argument are premuted along with the phenotypic status when permutation testing is performed; for this reason, this argument should include phenotypic control variables such as age and gender. This argument may be given multiple times.

--const-covariates string

Provides a comma-separated list of traits to include as covariates in the model. When using permutation testing, these covariates are NOT permutes along with the phenotypic status, and should include genotypic control variables such as principal components. Note that when not using permutation testing, this argument and the **--covariates** argument above are equivalent. This argument may be specified multiple times.

--outcome string

Provides a comma-separated list of traits to use as outcomes for the regression models. The generated models will be tested once for every trait given in this argument. This argument may be given multiple times. If this is not provided, PLATO will use the phenotypic status loaded with the data in the load-data command.

--encoding enum (=additive)

Specifies the encoding of the markers to be used in the models. The regression models used by PLATO assume a biallelic marker, with a given referent allele. In the polyallelic setting, all non-referent alleles are grouped into a single category, which we refer to as the “alternate” or “encoded” allele. Below are the valid encodings along with a brief description of each.

- **Additive**
  The classic GWAS encoding: Homozygous referent is “0”, heterozygous is “1”, and homozygous alternate is “2”.
- **Dominant**
  Homozygous referent is “0”, both heterozygous and homozygous alternate are “1”.
- **Recessive**
  Homozygous referent and heterozygous are “0”, while homozygous alternate is “1”.

- **Codominant**
  In this encoding, each marker uses two variables as a dummy encoding of a categorical variable. The “Het” variable is 1 only when the marker is heterozygous, and the “Hom” variable is 1 only when the marker is homozygous alternate. Note that this encoding incurs an additional degree of freedom for each marker used, and interactions also incur extra degrees of freedom.
- **Weighted**
  This encoding is a hybrid between the traditional encodings and the codominant encoding. For each marker, the result from a univariate model (with appropriate covariates) is used to determine an encoding from marker state to the set {0, x, 1}, where x is chosen such that the model with the encoded allele is identical to the codominant model. Then, this encoded allele is used in the multivariate models. Note that in the univariate and non-interaction case, this encoding is identical to the codominant encoding, but in the case of interactions, incurs fewer degrees of freedom.

--phewas

If given, PLATO will use all traits not explicity excluded or used as covariates as outcomes in a PheWAS. Note that this option is incompatible with the “--outcome” argument.

--correction enum

A comma-separated list of multiple test correction strategies to use in reporting results from the regression. Currently implemented strategies are limited to Bonferroni and FDR.

--output string (=output.txt)

The filename of the output of the regression. The output of the regression is given as a column-separated file, with one line per model tested, sorted by model p-value:

- Outcome name (if not using the default phenotypic status).
- IDs for the variables of interest.
- For any markers, allele and frequency of the encoded allele, calculated for the individual model. Note that his value may be different for the same marker in different models because of different patterns of missingness.
- The number of missing samples for the model.
- Convergence status (if using the logistic command)
- The coefficient, standard error and p-value for each variable. If testing interactions, the coefficients will be displayed for both the reduced model (main effects only) and the full model (main effects + interaction terms).
- The overall p-value of the model(s). Depending on the arguments used, there may be up to 3 distinct p-values for the model. When interactions are used, the overall p-value is the likelihood of significance when comparing the full model (covariates + main effects + interactions) to the reduced model (covariates and main effects only). The full and reduced model p-values are the likelihood of significance when comparing the respective model to one including covariates only. Without interactions, there is only a single p-value that is calculated comparing the model with variables of interest and covariates to one including only covariates.

- The corrected p-value(s) of the models. The corrected p-values are calculated based on the overall model p-value (see above for the meaning).

--separator string (=<TAB>)

The string to use for separating columns in the regression output file.

--show-univariate

If this argument is provided while examining multivariate models (with or without interaction terms), PLATO will also perform the appropriate regression using each variable alone and report the results. Note that the univariate regressions are not necessarily nested models, so the number of missing samples reported for each model may not correspond to the univariate results.

--thresh float (=1)

Gives a maximum p-value for printing models. This can be useful for showing only the most significant models in an analysis. Note that any multiple test correction strategies given will accurately correct for all tests performed, not just those pritned.

--threads int (=1)

The number of parallel processes to use in a regression in order to speed runtime. Setting this value to 0 disables threading, which can be useful in debugging PLATO.

--lowmem

If this argument is given, PLATO will use temporary files to output some of the results in an attempt to save memory. This can be a useful option when testing millions of models, but it may increase runtime due to the slow disk speeds when compared to memory.

load-data

This function loads genotypic data into PLATO for further processing or during concordance checking. Files are loaded from traditional genotype file formats including PLINK and Beagle. When parsing options for this command, PLATO will load the first set of files that is complete, in the following priority order (also the order of the commands listed in the help documentation):

1. PLINK files (ped/map)
2. Binary PLINK files (bed/bim/fam)
3. Transposed PLINK files (tped/tfam)
4. Long-format PLINK files (lgen/map/fam)
5. Beagle files
6. VCF Files

Any options that are given in the section of commands that do not apply to the loaded files will be ignored, potentially without warning. As an example, in the argument combination "--file myPlink --bgl-poly", the"--bgl-poly" argument will have no effect, as PLAT will load the PLINK files myPlink.ped and myPlink.map.

--file string

Gives a filename base for reading ped/map files. By default, PLATO will load the files *arg*.ped and *arg*.map, where *arg* is the given parameter to the “--file” argument. This can be overridden by the “--ped” and “--map” arguments.

--ped string

Specifies a PED file to load. Overrides the “--file” argument.

--map string

Specifies a MAP file to load. Overrides the “--file” and “--lfile” arguments.

--bfile string

Gives a filename base for reading bed/bim/fam files. By default, PLATO will load the files *arg*.bed *arg*.bim and *arg*.fam, where *arg* is the parameter to the “--bfile” argument. This can be overridden by the “--bed”, “--bim” and “--fam” arguments.

--bed string

Specifies a BED file to load. Overrides the “--bfile” argument.

--bim string

Specifies a BIM file to load. Overrides the “--bfile” argument.

--fam string

Specifies a FAM file to load. Overrides the “--bfile” and “--lfile” arguments.

--tfile string

Gives a filename base for reading tped/tfam files. By default, PLATO will load the files *arg*.tped and *arg*.tfam, where *arg* is the parameter given to the “--tfile” argument. This can be overridden by the “--tped” and “--tfam” arguments.

--tped string

Specifies a TPED file to load. Overrides the “--tfile” argument.

--tfam string

Specifies a TFAM file to load. Overrides the “--tfile” argument.

--lfile string

Gives a filename base for reading lgen/map/fam files. By default, PLATO will load the files *arg*.lgen, *arg*.map and *arg*.fam, where *arg* is the parameter to the "--lfile" argument. This can be overridden by the "--lgen", "--map", and "--fam" arguments.

--lgen string

Specifies an LGEN file to load. Overrides the "--lfile" argument.

--no-sex

If given, indicates that the PED file (or equivalent, in the case of binary/transposed filesets) does not contain gender information (typically column 5)

--no-parents

If given, indicates that the PED file (or equivalent) does not contain information about parental IDs (typically, columns 3 and 4).

--no-fid

If given, indicates that the PED file (or equivalent) does not contain family ID information (typically, column 1). Internally, the ID used to identify the sample will be the Individual ID repeated.

--no-pheno

If given, indicates that the PED file (or equivalent) does not contain phenotypic information (typically, column 6)

--map3

If given, indicates that the MAP file (or equivalent) does not contain genetic distance information (typically, column 3). Note that as of PLATO 2.0, genetic distance is ignored even if given.

--map-ref

If given, indicates that the MAP file (or equivalent) provides the referent allele in the column following the position information (typically, column 5). Note that in the case of reading a BIM file, this is implicitly selected.

--map-alt

If given, indicates that the MAP file (or equivalent) provides the alternate allele in the column following the referent allele (typically, column 6). Note that in the case of reading a BIM file, this is implicitly selected.

--control0

If given, indicates that the case/control phenotypic information from the PED file is encoded as 0/1, where 0 indicates a control and 1 indicates a case. By default, PLATO assumes a 1/2 encoding, where 1 indicates a control and 2 indicates a case, with 0 representing a missing value.

--quant

If given, indicates that the phenotypic information in the PED file is quantitative in nature. Note that the case/control status of all samples in this case will be unknown.

--beagle-prefix string

Gives the prefix for loading a set of Beagle files. By default, PLATO assumes a naming convention of *prefix.chrom.suffix*, where *prefix* is the parameter given to this argument, and *suffix* is the parameter given to "--beagle-suffix" or "--marker-suffix" for the genotype or marker file, respectively. Additionally, *chrom* is a chromosome string, and it is assumed that each genotype file has a corresponding marker file for a given chromosome.

--beagle-suffix string (=bgl)

Gives the suffix of the genotype files in a Beagle fileset. See "--beagle-prefix" for the naming convention. Files compressed with either gzip or bzip2 with an appropriate extension (.z or .gz for gzip and .bz for bzip2) will be automatically discovered and decompressed without the need to specify the compression extension.

--marker-suffix string (=markers)

Gives the suffix of the marker files for a chromosme in a Beagle fileset. See "--beagle-prefix" for the naming convention. Files compressed with either gzip or bzip2 with an appropriate extension (.z or .gz for gzip and .bz for bzip2) will be automatically discovered and decompressed without the need to specify the compression extension.

--beagle-files string

A comma-separated list of Beagle genotype files to load. This argument allows the user to load Beagle files that do not adhere to the convention assumed by PLATO, and will override any values given by "--beagle-prefix" or the suffix arguments. This argument must be using in conjunction with "--marker-files" and "--beagle-chroms", and filenames must be given exactly, but automatic decompression will still be performed given the appropriate extensions.

--marker-files string

A comma-separated list of Beagle marker files to load. Note that when using this option, there must be exactly as many marker files as genotype files and chromosomes given with the "--beagle-files" and "--beagle-chroms" options. As with "--beagle-files", filenames must be given exactly, but automatic decompression will still be performed given the appropriate extensions.

--beagle-chroms string

A comma-separated list of chromosomes that correspond to the files given by "--beagle-files" and "--beagle-markers". Note that all 3 arguments must have the same number of files / chromosomes.

--trio

If given, indicates that the data in the Beagle genotype files is trio data, with the convention that the data is listed in the order of "mother, father, child". Note that although the notation "mother" and "father" are used, no gender information is assumed. If the number of samples given in the genotype file is not a multiple of 3, PLATO will exit with an error. This argument is incompatible with the "--pair" argument.

--pair

If given, indicates that the data in the Beagle genotype files is pair data, with the convention that the data is listed in the order of "mother, child". Note that although the notation "mother" is used, no gender information is assumed. If the number of samples given in the genotype file is not a multiple of 2, PLATO will exit with an error. This argument is incompatible with the "--trio" argument.

--bgl-phased

If given, indicates that the data in the Beagle genotype files is phased. Note that biallelic phased data will occupy twice as much memory as biallelic unphased data. By default, PLATO assumes that Beagle files are biallelic and unphased.

--bgl-poly

If given, indicates that the data in the Beagle genotype file is polyallelic. **IMPORTANT NOTE:** Polyallelic data (phased or unphased) will occupy eight times the memory as biallelic unphased data and four times the memory of phased biallelic data.

--beagle-missing string (=0)

Gives the string to be interpreted as a missing allele when loading Beagle genotype files.

--vcf-file string

Gives the name of the VCF file to be loaded. If the VCF file is compressed, PLATO will automatically uncompress the data provided that the extension is appropriate (i.e. '.gz' for a (b)gzipped file or '.bz' for a bzipped file). Currently, PLATO does not use any associated index files.

--no-filter-marker

By default, PLATO will not load any marker that has a FILTER annotation other than "PASS" or ".". By setting the option, PLATO will load all markers without regard to the filter status.

--no-filter-geno

By default, PLATO will set any genotype that has a "FT" annotation other than "PASS" to missing. Including this option will cuase PLATO to load all genotype calls without regard to genotype filter annotation status.

--vcf-phased

Setting this option tells PLATO that the data in the VCF file is phased. Note that PLATO will warn the user if the phasing in the VCF file is inconsistent with the options provided.

--vcf-poly

Setting this option tells PLATO that the data sin the VCF file is polyallelic. If the VCF file is truly polyallelic and this option is not provided, PLATO will treat all non-referent alleles as the first alternate allele. Additionally, PLATO will warn if polyallelic markers are detected and this option is not provided.

load-trait

This command loads numeric data associated with samples into PLATO for further analysis. Note that this command should not be used to load numeric data that is categorical in nature (for example, numbers that represent sites, race or ethnicity). Examples of numeric data that can be loaded with this command include age, lab measurements, or quantitative phenotypic measurements (such as height, weight, BMI, blood pressure, etc.).

--file string

Gives the filename of the trait file to load. A trait file is a whitespace-delimited file with each row representing an independent sample. The first two columns give the Family ID and the Individual ID for the sample, and subsequent columns provide the numeric value. Empty lines, or lines beginning with a pound sign (#) are ignored, and the first line of the file is considered the header line, and the values given are the trait IDs that will be available for use in subsequent steps. An example of a trait file is below:

```
# This line is a comment, and ignored
FID   IID   Age   BMI
# Note the above line, which loads 2 traits, 'Age' and 'BMI'
1     1     25    22.4
2     2     34    21.7
# See --ignore-error for how to handle 'NA'
3     3     NA    29.6
# See --missing for how to handle '-9999'
4     4     52    -9999
                    ... (Further lines truncated) ...
```

--missing string

Gives a string value to treat as a missing value. This is especially helpful when missing data is represented by a valid number that is nonsensical in the context of the variables being loaded (e.g. "-9999" as a value for BMI in the above example).

--no-fid

Indicates that the first column (Family ID) is missing from the file. Internally, PLATO will attempt to find the appropriate sample by using the Individual ID repeated as the Sample ID.

--ignore-error

Allow non-numeric values in a trait file and treat those unparseable values as missing. Typically, PLATO will exit with an error if it encounters a non-numeric value; this is especially helpful when non-numeric values are used to represent missing data (such as 'NA' or '.').

--extra-samples

If samples are found in the trait file that are not present in the currently loaded, simply ignore the line. By default, PLATO will exit with an error if a sample is present in a trait file but not present in the previously loaded genotype data. Note that samples that have previously been marked as disabled are still considered present in the dataset. This option is especially helpful when loading phenotype data which may have been collected on more samples that the corresponding genotype data.

--dummy-samples

Similar to the "--extra-samples" option above, if this is provided, PLATO will create a sample with completely missing genotype data for any samples found in a trait file but not found in the previously loaded genotype data. This option is helpful for performing an EWAS or ExE analysis in which the user may not have genotypic data.

--require-complete

If this option is given, PLATO will require an entry in the trait file for all enabled samples currently loaded into memory. This can be a helpful check to ensure that the files are complete and correspond to the same IDs.

logistic

This command performs logistic regression on the data previously loaded into PLATO. The logistic regression is performed using a Newton-Rhapson iteratively reweighted least squares algorithm. Many of the options for this command are also applicable to the linear command. For the sake of compactness of the manual, the options documented below are those that are specific to this command.

NOTE: This command is MPI-enabled.

--odds-ratio

If given, PLATO will display the odds ratios for each coefficient rather than the raw coefficient returned by the logistic regression.

--max-iterations int (=30)

Gives the maximum number of least-squares steps allowed before declaring nonconvergence of the method. Increasing this number will make the regression more robust, but may drastically increase the runtime, especially in the cases where convergence using a simple logistic regression is impossible.

output-beagle

This command will output all currently enabled markers and samples in Beagle file format (see http://faculty.washington.edu/browning/beagle/beagle_3.3.2_31Oct11.pdf for a full description of the file formats. Note that phasing and polyallelic data is automatically detected according to the data types loaded into PLATO.

PLATO will output the genotype and corresponding marker file for each chromosome that contains data. The files will be named according to the pattern <prefix>.<chrom>.<suffix>, where prefix and suffix are user-specifiable strings using the arguments below.

--prefix string (=output)

This is the prefix to use for all beagle files.

--suffix string (=bgl)

This is the suffix to use for all genotype files. Note that currently, PLATO does not support compressed Beagle file output, but this is a feature we hope to add shortly.

--marker-suffix string (=markers)

This is the suffix to use for all marker files. As with the genotype files, PLATO does not support output of compressed Beagle format directly.

--incl-traits

When this argument is given, PLATO will include the affection status as well as all currently loaded traits in the genotype files that are output. These lines correspond to "A" and "T" output lines in the Beagle specification.

--pair

Indicates that the data to be output is parent-child data, which affects the ordering of the samples. Note that PLATO will exit with an error if the data loaded does not conform to this structure.

--trio

Indicates that the data to be printed is trio data (two parents, one child). Note that PLATO will exit with an error if the data loaded does not conform to this structure.

--missing string (=0)

Gives the string to use to indicate the presence of a missing allele.

output-bed

This command outputs the currently loaded genotype data in binary PLINK (bed/bim/fam) format (see http://pngu.mgh.harvard.edu/~purcell/plink/binary.shtml for a complete description of the format).

One note in the difference between PLATO and PLINK is that PLATO does not recode the alleles unless explicitly requested in recode-alleles. Additionally, when automatic recoding of alleles is requested, the major (and referent) allele will be printed in column 5 of the bim file and the minor (coded) allele will be printed in column 6, which is opposite of the behavior of PLINK. Despite this difference in behavior, both programs will print files that represent the same information and should be 100% concordant.

Important Note: currently, PLATO ignores any genetic distance (column 3 of the map file) given when loading PLINK genotype files. When printing bim files, column 4 will ALWAYS be 0.

--file [-f] srting (=plato)

This argument gives the base filename of the bed/bim/bam files. By default, PLATO will output *arg*.bed, *arg*.bim, and *arg*.fam files, where *arg* is the given parameter for this argument. This behavior can be overridden on an individual file basis using the "--bed", "--bim", or "--fam" arguments.

--bed string

Provides an explicit filename for the bed file, overriding the "--file" argument.

--bim string

Provides an explicit filename for the bim file, overriding the "--file" argument.

--fam string

Provides an explicit filename for the fam file, overriding the "--file" argument.

--individual-major

Outputs the bed file in individual-major mode (default is SNP-major mode).

output-eigenstrat

This command outputs the currently enabled genotype data in Eigenstrat file format (see http://genepath.med.harvard.edu/~reich/InputFileFormats.htm for a full description of the format).

--file [-f] string (=plato)

This argument gives the base filename of the Eigenstrat files. By default, PLATO will output *arg*.geno, *arg*.snp, and *arg*.indiv for the genotype, SNP and individual files, respectively. This behavior can be overridden on an individual file basis using the "--geno", "--snp", and "--indiv" arguments.

--genotype string

Provides a filename for the genotype file, overriding the "--file" argument.

--snp string

Provides a filename for the SNP file, overriding the "--file" argument.

--indiv string

Provides a filename for the individual file, overriding the "--file" argument.

output-ped

This command prints all currently enabled genotype data in traditional PLINK (ped/map) format (see http://pngu.mgh.harvard.edu/~purcell/plink/data.shtml for a complete description of the file format).

Important Note: currently, PLATO ignores any genetic distance (column 3 of the map file) given when loading PLINK genotype files. When printing map files, column 4 will ALWAYS be 0.

--file [-f] string (=plato)

Provides a base for the filenames to print the data to. By default, PLATO will output *arg*.ped and *arg*.map, where *arg* is the parameter passed to this argument. This behavior can be overridden on an individual file basis using the "--ped" and "--map" arguments.

--ped

Provides a filename for the ped file, overriding the "--file" argument.

--map

Provides a filename for the map file, overriding the "--map" argument.

output-tped

This command prints all currently enabled genotype data in transposed PLINK (tped/tfam) format (see http://pngu.mgh.harvard.edu/~purcell/plink/data.shtml#tr for a complete description of the file format).

Important Note: currently, PLATO ignores any genetic distance (column 3 of the map file) given when loading PLINK genotype files. When printing tped files, column 4 will ALWAYS be 0.

--file [-f] string

Provides a base filenamefor the transposed PLINK files. By default, PLATO will output *arg*.tped and *arg*.tfam. This behavior can be overridden on an individual file basis using the "--tped" and "--tfam" arguments.

--tped string

Provides a filename for the tped file, overriding the "--file" argument.

--tfam string

Provides a filename for the tfam file, overriding the "--file" argument.

recode-alleles

This command changes the referent allele for the markers currently loaded by previous commands. This is helpful in order to correctly interpret the direction of effect of the coefficients in regression models. Note that by changing the referent allele, the user will implicitly change the encoded allele.

--file string

Provide a two-column whitespace-separated file of marker IDs and the desired referent allele. Markers can also be identified by chromosome and base pair location by using a colon (:) to separate the two. Empty lines and lines beginning with a pound sign (#) are ignored. Note that if an allele is given but not present for a marker, a warning will be issued, but the marker will remain unchanged. Below is an example of this file format:

```
# Example recoding allele file
rs23562           A
3:237630          T
chr11:9362856     C
                  ... (Additional lines truncated) ...
```

--input-map

If this argument is given, we assume that the file provided above is in extended map format, with an additional column providing the referent allele. In this case, markers are found using the provided ID, or the chromosome and base pair position provided in the map file.

--map3

If this argument is provided, both input and output map files will not contain the genetic distance column (fourth column in a traditional map file).

--auto

If given, PLATO will set the referent allele as the most common allele present in the dataset. In biallelic data, this has the effect of making the minor allele the coded allele. If the "--file" argument is also given, this argument is superseded by those markers found in the recoding file.

--out string

Provides a filename to print the marker ID and the referent allele after this step. This is particularly useful when automatically detecting the referent allele and then seeking replication in another dataset, as this will ensure that the coded allele is identical in both analyses.

--output-map

Prints the output above in extended map format. This may be useful as part of a pipeline including other tools.

regress-auto

This command will perform either a linear or logistic regression based on the phenotype provided to PLATO. If there are exactly two non-missing phenotypes, this command will perform a logistic regression, as used in the logistic command. If there are more than two unique non-missing phenotypes, this command will perform a linear regression, as in the linear command. The output of the regression will include a column indicating the type of regression performed for each model in addition to all of the output columns of the logistic command. For linear regression models, the “Converged” column will always be 1, as all linear regressions are not iterative, and so they will always “converge” in one step.

Note that this command may be dangerous, as it can be difficult to compare linear and logistic regression results in a statistically meaningful way.

--linear string

This option takes a comma-separated list of trait variables for which PLATO should perform linear regression, regardless of the number of non-missing phenotypes. The variables provided to this option and “--logistic” below must be mutually exclusive and given in the “--outcome” option if “--phewas” is not provided. This option may be specified more than once.

--logistic string

This option takes a comma-separated list of trait variables for which PLATO should perform logistic regression, regardless of the number of phenotypes. The variables provided to this option and “--linear” above must be mutually exclusive and given in the “--outcome” option if “--phewas” is not provided. This option may be specified more than once.

Advanced Topics

Pre-Filtering Data

In order to allow analysis on extremely large datasets, PLATO provides the option to only load a portion of the dataset provided into memory. Note that unlike traditional filters like filter-maf or filter-sample-call, if a marker or sample is excluded during the pre-filtering stage, it cannot be included in any model or later analysis step. While these options can reduce the memory footprint of PLATO, they will not reduce the time to load data, as the entire dataset must be read in order to ensure correct loading of all data.

--chrom string

This argument gives a comma-separated list of chromosomes to load data for. If not given, PLATO will load data for all chromosomes.

--bp-window srting

This argument gives a comma-separated list of base pair windows to include in the data loading. The windows must be formatted as "#-#", providing a lower and upper bound (inclusive) of positions to load data. Note that either the upper or lower bound may be omitted, in which case, the bound will be interpreted to mean either "0" or "max int", respectively. Caution should be taken when including base pair windows with multiple chromosomes, as the windows will apply to all chromosomes being loaded.

--incl-marker string

This command gives a comma-separated list of marker IDs to include. This can be useful when extracting a very small number of SNPs from a large dataset. Note that marker IDs can be the traditional RSID, or optionally a "chrom:basepair" string. This option may be specified more than once.

--excl-marker string

This command gives a comma-separated list of marker IDs to exclude. This can be useful when excluding a small number of markers from a dataset. Note that if a marker is both explicitly included and explicitly excluded, the marker will NOT be loaded. This option may be specified more than once.

--incl-marker-fn string

This command gives a comma-separated list of files containing a list of marker IDs to include, one per line. This can be helpful for including a large number of markers for analysis, as an example from LD pruning. This option may be provided more than once.

--excl-marker-fn string

This command gives a comma-separated list of files containing marker IDs to exclude, one per line. This option can be helpful when excluding a large number of markers, as an example when LD pruning. Again, note that if a marker is explicitly included and explicitly excluded, the marker will NOT be loaded.

--incl-sample string

This command gives a comma-separated list of sample IDs to include for analysis. This can be helpful in extracting a small subset from a large dataset. Note that if samples are identified by both an IID and an FID, the sample ID should include both, starting with the FID, and they should be whitespace separated. For this reason, the user may need to enclose this argument in quotes. This option may be specified more than once.

--excl-sample string

This command gives a comma-separated list of sample IDs to exclude in any subsequent analysis. This can be helpful in excluding problematic samples from analysis, such as cryptically related samples. As above, care must be taken when using sampled identified by both FID and IID, and if a sample is both explicitly included and explicitly excluded, the sample will NOT be loaded. This option may be specified more than once.

Permutation Testing

PLATO offers permutation testing for calculation of the p-value of a model in addition to the parametric p-value calculation inherent with the regression models. The p-value for a given model is defined to be the fraction of times any permuted model has a lower p-value than the parametrically calculated p-value of the unpermuted model. Note that the permuted p-value should still be subject to multiple test correction, and the granularity of the adjusted permuted p-value will be determined by the number of permutations. Thus, if a user ran 1,000 permutations, the smallest nonzero adjusted permuted p-value will be 0.001.

The following options control the behavior and output of the permutations run by PLATO.

--permutations int (=0)

This argument determines the number of permutations per model to run. Note that the runtime and memory requirements will increase dramatically with this parameter. At a minimum, each permutation will take the same amount of time as a single model and will require approximately 12 bytes of memory (in **--lowmem** mode). For a modest GWAS consisting of 500,000 markers with 1,000 permutations, PLATO will need approximately 6 extra GB of RAM and will take 1,000 times as long to execute.

--permu-seed int

This argument allows for repeatability of results by specifying the initial seed for the random number generator used to initialize the permutations. If no seed is given, one will be generated at random and reported to the user.

--permu-thresh float (=0.05)

PLATO will print detailed permutation results for all permuted models whose p-value falls below this threshold. Be aware that increasing this threshold can dramatically increase the storage requirements.

--permu-detail-fn string (=permutation-detail.txt)

This is the name of the file that PLATO will print the detailed permutation output to. To disable this output, set this argument equal to the empty string (“”).

--permu-pval-fn string (=permutation-pval.txt)

PLATO will print all p-values of all permuted models in this file, one per line. The p-values printed will be sorted, with the following "special" p-values as flags with the following interpretation:

- **2**: Calculation of the permuted model failed
- **3**: Could not run permuted model (typically, too many missing values)

--permu-run-full

By default, PLATO will only run enough models in the permutation data to generate an appropriate p-value. If the user specifies this option, PLATO will run all submodels during a permutation. This option only has an effect when permuting interaction models, and in that case, specifying this option will increase the runtime by 33%.

Model Generation for Regression

PLATO can test a variety of models in the regression procedures. Typically, models are generated by iterating over the dataset and running a separate regression for each model generated. The manner in which the user iterates over the dataset will determine the type of analysis being run, with possibilities being GWAS, EWAS, GxE, GxG, and many others. Alternatively, the user can individually specify the models to be run, which can be helpful in the case of a targeted subset of models.

The options below are used by the regression options, linear and logistic. The Model Generation Summary section provides a good summary of the combination of these options and the resultant analysis that can be run using PLATO.

--models string

This parameter takes a comma-separated list of files containing models to test. Each line in the file is assumed to be a whitespace delimited list of variables to include in the model and PLATO assumes that the structure of all models contained in the model files are identical to the first. For each variable, PLATO attempts to identify by marker ID, marker chromosome and position (separated by :), then trait (in that order). Empty lines and lines beginning with a pound sign (#) are ignored. An example of a model file is below:

```
# This is a comment, and ignored
# All models have 2 markers + a trait
rs122483    chr1:342864       BMI
chr2:86302  chr4:47290        LDL
# Note that order of the variables doesn't matter
LDL         rs347290          3:12386
# Also, RSIDs are case insensitive and "chr" is optional for a chrom.
Rs122483    BMI               4:47290
```

Note that this argument can be given multiple times, and if given, will override any automated model generation.

--exclude-markers

If given, automatically generated models will not include markers. This is useful for an EWAS or an

ExE analysis.

--use-traits

If given, will include the traits not listed as covariates in the models to be generated.

--pairwise

Generate exhaustive pairwise models.

--incl-traits string

Gives a comma-separated list of traits to include when generating models. This argument may be provided multiple times. Note that "--excl-traits" takes precedence over this argument, so if a trait is both included and excluded, the given trait will be excluded from analysis. This is especially helpful when used in conjunction with the "--one-sided" argument.

--excl-traits string

Gives a comma-separated list of traits to exclude from analysis when generating models. This argument may be given multiple times and supersedes "--incl-traits".

--incl-markers string

Gives a comma-separated list of markers to include in the model generation. This argument may be given multiple times, and is especially helpful when used in conjunction with the "--one-sided" argument.

--one-sided

When used in conjunction with the "--pairwise" argument, will instruct PLATO to generate models with one marker (or trait, if "--exclude-markers" was given) chosen from the provided list of markers (or traits), and the other being chosen from all markers (or traits) not explicitly excluded.

Model Generation Summary

Because the options for generating PLATO models can be somewhat overwhelming, the table below lists some of the options available and what the corresponding analysis. In the table, options are listed by either "Y", "N", or "*", which means that the option was given, not given, or either given or not:

--exclude-markers	--use-traits	--pairwise	--one-sided	Analysis
N	N	N	N	GWAS
N	Y	N	N	GxE
Y	Y	N	N	EWAS
N	N	Y	N	GxG
Y	Y	Y	N	ExE

--exclude-markers	--use-traits	--pairwise	--one-sided	Analysis
N	Y	Y	N	GxGxE
N	N	Y	Y	Focused GxG
Y	Y	Y	Y	Focused ExE
Y	Y	Y	Y	Focused GxGxE
Y	N	*	*	Error
*	*	N	Y	Error