
Using **PLINK** with R plugins on TARDIS

This document will describe how to utilize the **PLINK** software with R plugins on TARDIS. We'll begin by describing how to ensure the essential software is available to your environment, subsequently installing then testing the necessary **Rserve** in the background. Then a scheme for partitioning large SNP sets to facilitate parallelization will be described. Finally, we will produce a set of batch scripts to queue and use the Moab scheduling software to run multiple **PLINK** jobs in parallel for analyzing SNP data across a chromosome.

Updating the software environment

First check to see if the most recent version of R is loaded into your software environment:

```
which R && R --version
```

Here is sample output from this command:

```
[rduncan-emory@tardis-6 ~]$ which R && R --version
/usr/local/packages/R/2.13.1/gcc-4.4.5/bin/R
R version 2.13.1 (2011-07-08)
Copyright (C) 2011 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: x86_64-unknown-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under the terms of the
GNU General Public License version 2.
For more information about these matters see
http://www.gnu.org/licenses/.
```

The version of R loaded into the software environment here (2.13.1) is rather old. We can remove that version from the software stack then add the latest version using the **module** command:

```
module rm R
module add R/3.0.1
```

Similarly, check to see if **PLINK** is available:

```
which plink
```

```
[rduncan-emory@tardis-6 ~]$ which plink
/usr/bin/which: no plink in (/usr/local/packages/R/3.0.1//bin/:/usr/local/packages/rstudio/0.97.551/R-3.0.1/bin:...)
```

If not, then add it to the software environment:

```
module add plink
```

PLINK makes calls to R through the Rserve daemon which will be considered next.

Installing and executing Rserve

In this section we will install Rserve as a package to the local directory tree. We'll then briefly start it as a background process to test its readiness.

With only user-level privileges, we must install Rserve and other needed packages locally. First, if they doesn't already exist, create a set of directories for managing local R package installations and code sources:

```
mkdir -p ~/R/library ~/R/src
```

The latest version of Rserve should be found at the CRAN site:

<http://cran.r-project.org/web/packages/Rserve/index.html>.

Using the link address to the package source code there, download Rserve into the local source directory:

```
wget http://cran.r-project.org/src/contrib/Rserve_1.7-3.tar.gz -P ~/R/src/
```

Now perform the local package installation of Rserve:

```
R CMD INSTALL -l ~/R/library ~/R/src/Rserve_1.7-3.tar.gz
```

Running Rserve in the background

In order for **PLINK** to make calls to R, Rserve will need to be running as a background process. A short R test script to test this capability follows:

```
#-----  
# run-Rserve.R  
#-----  
.libPaths("~/R/library")  # where local packages are found  
library(Rserve)           # load Rserve package  
Rserve(args="--no-save")  # run Rserve in background
```

Save this as **run-Rserve.R** then execute it from the command line in the usual way:

```
R --slave < run-Rserve.R
```

To verify that Rserve is now running in the background, run **grep** against the process report:

```
ps -ef | grep Rserve
```

Here is a sample output from this command:

```
[rduncan-emory@tardis-6 ~]$ ps -ef | grep Rserve  
480020  24171    1  0 16:14 ?        00:00:00 /nv/het1/rduncan-emory/R/library/Rserve/libs//Rserve --no-save  
480020  24304 24007  0 16:15 pts/2    00:00:00 grep Rserve
```

The first column is the user ID. The second column contains the process ID (PID), which will be important later. In this case, **grep** returned two processes: the actual **grep** call itself (PID 24304) and the running Rserve (PID 24171). After we have finished using Rserve, this latter PID can be used to kill the Rserve daemon during clean-up.

Since we'll call Rserve from the R scripts used with **PLINK**, let us kill this test Rserve daemon. A safe way to execute the kill command is

```
killall -i --exact Rserve
```

The option **-i** makes this command interactive, i.e., forces you to answer yes/no questions before proceeding, while the **--exact** option matches processes that have the exact word 'Rserve' in the name.

Alternatively, if you are sure of the PID for the Rserve process, it can be killed using the command:

```
kill -s SIGKILL PID
```

where *PID* is replaced with the actual process ID number, e.g., in this case 24171.

Note that this extra step of terminating the background Rserve process will not be necessary during the production run: the scheduler will handle that detail.

Now that Rserve is installed and working we are set to run **PLINK** with R plugins. With potentially tens- or hundreds-of-thousands of SNPs across the genome to include in the **PLINK** analysis, it will be advantageous to partition the SNPs into smaller sets that are appropriate for parallelization. In the next section, we'll use Perl scripts to build a set of **PLINK** commands to analyze those partitioned subsets then produce a set of batch scripts that can be queued to the scheduler along with a master script to perform the actual queueing.

Partitioning SNP indices along a chromosome

A Perl script **partition-snps-by-chromosome.pl**, has been developed to call contiguously partition the SNP index set subsequently creating the corresponding **PLINK** command for each partition. Eventually, these commands will be executed separately by the scheduler. Using the **--help** option produces a more detailed description of this script and the options passed to it.

```
./partition-snps-by-chromosome.pl --help
```

To print a count of SNPs across a given chromosome, use the **--summary** option. In this example, we ask for a summary of chromosome 1 SNPs in file **AA.bim** with **--N=5** partitions:

```
./partition-snps-by-chromosome.pl --N=5 --bfile=AA --chr 1 --summary
```

```
SNP count in chromosome 1:  53395
SNP count in each partition: 10679
```

Thus, there are 53,395 SNPs along chromosome 1 in this data set and partitioning into five subsets yields 10,679 SNPs in each batch. The command to construct then store the set of corresponding **PLINK** command calls into a file named **cmds** is:

```
./partition-snps-by-chromosome.pl --N=5 --bfile=AA --out=BDI --covar=covariate.cov --Rplink=Rplink.R --silent --chr 1 > cmds
```

The output of this script is printed to the file **cmds** consisting of the list of **PLINK** commands for each partition preceded by a brief summary of SNP counts. Within each of those commands the **--snps** option passes to **PLINK** a beginning and ending SNP for each respective **PLINK** call. The output file looks like this:

```
plink --noweb --bfile AA --covar covariate.cov --R Rplink.R --chr 1 --silent --snps rs3094315-rs33950227 --out BDI_chr01_rs1
plink --noweb --bfile AA --covar covariate.cov --R Rplink.R --chr 1 --silent --snps rs33953680-rs7340081 --out BDI_chr01_rs2
plink --noweb --bfile AA --covar covariate.cov --R Rplink.R --chr 1 --silent --snps rs6697536-rs11265584 --out BDI_chr01_rs3
plink --noweb --bfile AA --covar covariate.cov --R Rplink.R --chr 1 --silent --snps rs12093075-rs600031 --out BDI_chr01_rs4
plink --noweb --bfile AA --covar covariate.cov --R Rplink.R --chr 1 --silent --snps rs6671923-rs12025760 --out BDI_chr01_rs5
```

Furthermore, we can run this for any other chromosomes of interest, appending to the file the resulting set of commands:

```
./partition-snps-by-chromosome.pl --N=5 --bfile=AA --out=BDI --covar=covariate.cov --Rplink=Rplink.R --silent --chr 2 >> cmds
```

```
./partition-snps-by-chromosome.pl --N=5 --bfile=AA --out=BDI --covar=covariate.cov --Rplink=Rplink.R --silent --chr 3 >> cmds
```

etc.

Note the difference in the use of '> **cmds**' in the first call to the script and '>> **cmds**' in subsequent calls. The former overwrites any old file contents or creates a new file with the output. The latter appends the outputs to the already existing file. Thus, the commands for all chromosomes can be maintained in a single file.

Obviously, a much finer partitioning would be better for taking advantage of parallelization on a big cluster, e.g., say **--N=100**. Consequently, with so many commands to queue, e.g., ~2000 such commands to cover all SNPs across every chromosome, this would be difficult to manage manually. Thus, another script, described next, will be used to produce the corresponding set of batch scripts while maintaining a master script containing a queueing command for each script.

Partitioning SNP indices across the genome

Another Perl script **partition-snps-genome.pl**, has been developed generate **PLINK** commands for a partitioning across the genome with a specified number of SNPs to each batch. This is accomplished by calling the previous script **partition-snps-by-chromosome.pl** for each chromosome in the genome. Eventually, the commands generated by this set of scripts will be executed separately by the scheduler. Using the **--help** option produces a more detailed description of this script and the options passed to it.

```
./partition-snps-genome.pl --help
```

To print a count of SNPs across a given chromosome, use the **--summary** option. In this example, we ask for a summary of chromosome 1 SNPs in file **AA.bim** with **--N=5** partitions:

```
./partition-snps-genome.pl --N=5 --bfile=AA --chr 1 --summary
```

Producing the queueing batch scripts

Now that the **PLINK** commands are constructed, the Perl script **make-plink-schedule.pl** can be used to (i) generate each batch script corresponding to each command and (ii) produce a master list commands for queueing by the scheduler.

The command might look like this:

```
./make-plink-schedule.pl --in=cmds --out=plink-qsubs --template=run-plink.template
```

The resulting schedule file is simply a shell script to qsub each batch:

```
#!/bin/sh
qsub BDI_chr01_rs1.sh
qsub BDI_chr01_rs2.sh
qsub BDI_chr01_rs3.sh
qsub BDI_chr01_rs4.sh
qsub BDI_chr01_rs5.sh
```

where each auto-generated batch file can be qsub-ed for a **PLINK** analysis of a specific partition:

```
# script to run PLINK via MOAB on TARDIS
#
#PBS -N BDI_chr01_rs1
#PBS -l nodes=1:ppn=1
#PBS -l pmem=8
#PBS -l walltime=12:00:00
#PBS -q tardis-6
#PBS -j oe
#PBS -o log/$PBS_JOBID.log
#PBS -V
#PBS -r y
#PBS -A WHAT-GOES-HERE?

echo "PLINK analyzing partition 1 of chromosome part of chromosome 1 on `/bin/hostname`"
plink --noweb --bfile AA --covar covariate.cov --R Rplink.R --chr 1 --silent --snps rs3094315-rs242050 --out BDI_chr01_rs1
```

Putting it all together

Thus the sequence of commands to perform a **PLINK** analysis of the SNPS across all chromosomes with **M=100** SNPs per batch may look like this:

```
./partition-snps-genome.pl --M=100 --bfile=AA --out=BDI --covar=covariate.cov --Rplink=Rplink.R --silent > cmds
./make-plink-schedule.pl --in=cmds --out=plink-qsubs --template=run-plink.template
./plink-qsubs
```

NB: plink-qsubs still untested

To do

fix account specification in billing line of PBS script

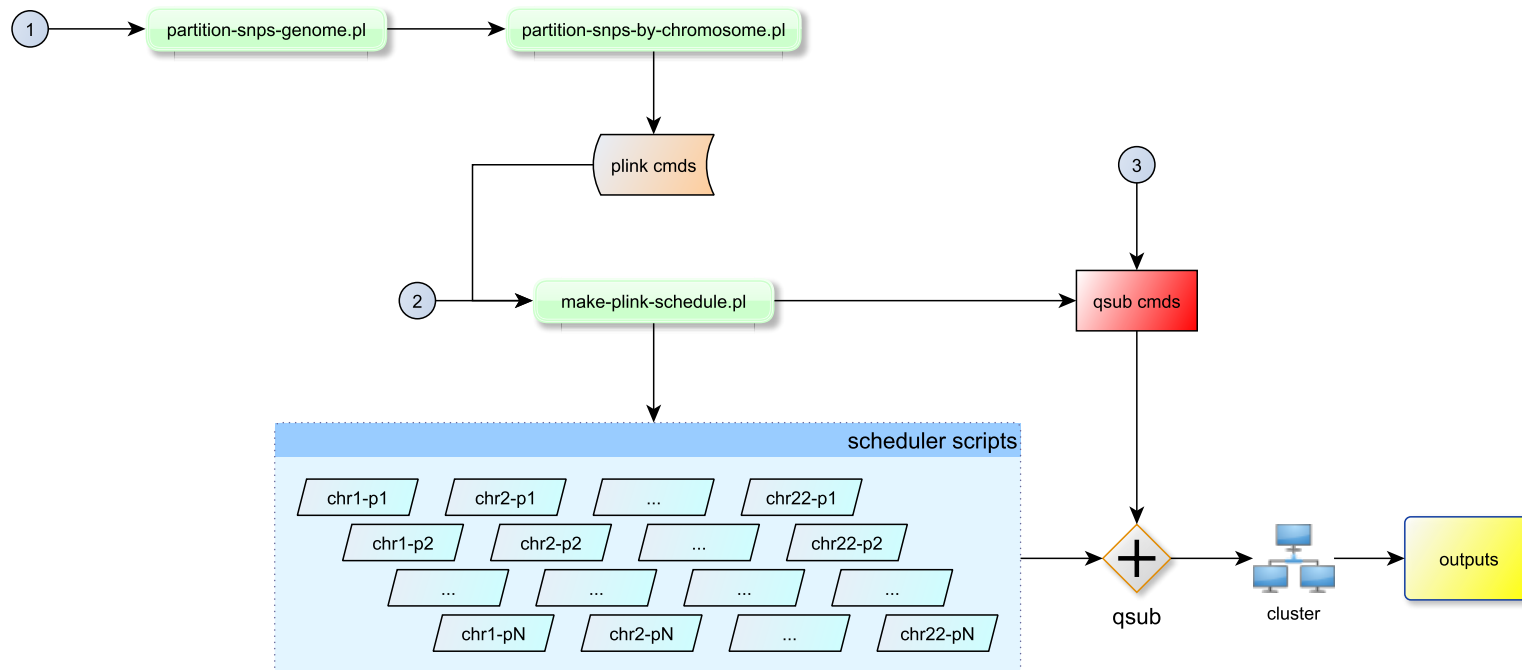


Figure 1 Sequence for generating **PLINK** scripts and a corresponding executable schedule script for SNPs partioned across the genome.