

# The Biological Repository (BioR) and BioRTools User Guide v2.2.x

---

By Daniel Quest, Mike Meiners, Patrick Duffy, Raymond Moore, The BioR Team, and the BioR users.

## Table of Contents:

### 1. Installation:

[Installing inside Mayo with access to the Research Computing Facility \(RCF\)](#)

[Overview](#)

[Steps](#)

[Installing the Biological Repository Catalogs](#)

[Installing on a Stand-Alone Server or Workstation](#)

[Installing BioR Tools from Source](#)

[Java Heap Size](#)

### 2. Overview

[Introduction](#)

[Data Modeling](#)

[BioR Catalog Shortcut](#)

[Finding out what is in a Catalog](#)

[Showing the Commands in BioR Toolkit](#)

[Pretty Print](#)

[Get all Variants in a Gene](#)

### 3. BioR Catalogs

[The BioR Catalog Format](#)

[Catalog Creation Details](#)

[Catalogs Available In BioR](#)

[Creating a New Catalog in Seconds](#)

### 4. Examples Matching Genomic Features

[Positional Matches Using Tabix](#)

[Annotating Variants with Genes that Overlap](#)

[Compressing output to enforce 1-1 semantics](#)

### 5. Expanded Genes (Xrefs)

[Indexing Catalogs](#)

[Looking Up Information about a Gene](#)

[Example of Walking Cross References](#)

[Generating an OMIM Disorder Report for a Set of rsIDs](#)

[Putting it all Together – Making a Genomic Feature Annotation Program](#)

### 6. Examples Matching Alleles (bior same variant)

[Putting it All Together Building an AF Pipeline](#)

### 7. Extracting Data with JSONPaths (bior drill)

### 8. Command Line Tools

### 9. Mixing In Scripts and Languages

[To find all overlapping genes that are not the same gene:](#)

### 10. Common Problems

[Handling VCF Files with VERY large headers](#)

[Large Memory Requirements](#)

[BioR exits with some error I don't understand](#)

## 11. Creating Catalogs

[Indexing your Samples](#)

[Creating Custom Catalogs](#)

[The Publication Process](#)

[Parsing and Converting the Data](#)

[Indexing the Data for Coordinate Based Search](#)

[Hints on Creating Indexes on Custom Catalogs](#)

[Use BioR to map SNP on rsID and find overlapping genes.](#)

[Case Study: Creating a Report that Maps rsIDs to Genes.](#)

## 12. Sun Grid Engine

[Multiple Cores](#)

[Virtual Memory](#)

[Resources for a Toolkit Pipeline](#)

# The Biological Repository (BioR) and BioRTools User Guide v 2.2.x

---

BioR is an annotation engine. Inside Mayo, it's primary use is to annotate human variation, but it is not limited to that – it is a general purpose genomic data integration tool that enables coordinate based searches and joins based on strings. BioR is like programming using lego blocks, each block may not be exactly what you want, but you can put the blocks together to create programs extremely rapidly. The component 'blocks' include all existing UNIX commands, stand alone tools (e.g. bedtools), and the bior\_toolkit. This user guide will help get you up to speed in how to use BioR in one document. Please note that BioR is a complex system, and you should have some experience with UNIX (especially pipes) before using BioR. BioR consists of two parts 1) the BioR toolkit which depends on Java (some commands also depend on SNPEFF and VEP) and 2) the BioR catalogs which are the data files used by the system. A BioR catalog is basically a BED-JSON hybrid file that is indexed using Tabix for coordinate based search and BioR's own indexing system for string matching based searches.

## 1. Installation:

### Installing on a server in the cloud

A Setup guide for installing on the cloud is coming soon!

- 1) Check the version of your Linux OS by typing 'uname -a' at the command line. if it says x86\_64, you have a 64bit system (most systems).
- 2) Based on the version of your OS, download Java directly from Oracle. We recommend JDK 1.7. (I usually download it to my home computer and then scp it up to the server using winscp or scp from the command prompt on a mac. This will result in a file in your home directory called jdk-7u45-linux-i586.tar.gz. Unzip Java and set your JAVA\_HOME.
- 3) Follow the section below for installing the catalogs on a stand-alone-server
- 4) Follow the section below for installing the toolkit on a stand-alone-server
- 5) Follow the section below for installing SNPEFF/VEP and other command line tools.

That is all that is needed to install BioR on the cloud.

### Installing inside Mayo with access to the Research Computing Facility (RCF)

If you have access to the RCF, you are in luck! We have already installed BioRTools for you, all you need to do is put it in your path. Here are the steps to do that:

#### Overview

The CLI is available through the **mayobiotools** utility. No software needs to be downloaded as it's already pre-installed. Make sure you select version 2.0 or greater.

#### Steps

1. login to an RCF submission node server (example: "ssh crick6.mayo.edu")
2. execute "mayobiotools"
3. scan the list of packages for "java"
4. type corresponding package number and press enter
5. select a version that is 1.6 or higher

6. scan the list of packages for "bior\_scripts"
7. type corresponding package number and press enter
8. select "2.2.0" version
9. quit mayobiotools and save changes
10. logout and log back into the RCF submission node server
11. BioR Command Line Client commands are now available
12. Try this from the command line: "bior\_vcf\_to\_tjson -h" if BioR is working you should see a help message.
13. To explore the bior scripts available on the command line type bior followed by a tab.

## Installing the Biological Repository Catalogs

On the RCF, no installation is needed. Catalogs can be found at \$BIOR\_CATALOG (\$bior in this documentation) If you are doing a stand alone server, download the catalog flat files and place them locally on your server in a similar directory structure. BioR Tools does not make any assumptions about the location of catalogs relative to each other, but it does assume that tabix indexes are in the same directory as the compressed catalog and that ID indices are in a folder called index in the same directory as the catalog. However, bior\_annotate does have a configuration file that will make that command not work if you don't change the configuration file or place the data repositories in the same location as we do at Mayo (or provide a symbolic link). We put the data here: \$BIOR\_CATALOG=/data5/bsi/catalogs/bior/v1. More details for installing the catalog structure properly on a stand alone server can be found in the next section "Installing on a Stand-Alone Server or Workstation" (next).

## Installing on a Stand-Alone Server or Workstation

BioR is written in Java, so in principle it will work on any machine, but it depends on some command line tools (e.g. SNPEFF, VEP) that are not so friendly. The development team has BioR working on both Macintosh and Linux. To install, first make sure first that Java 1.6+ is installed and on your path (Java 1.7 is preferred). Then download the BioR executable and place it in your path.

### Download Links:

You can download BIOR and Catalog datasources from <http://bioinformaticstools.mayo.edu/research/bior/>.

### Java Installation:

We prefer to use the official version of Java directly from Oracle (Sun). Use uname -a on your Linux machine to determine the version of Java that you should be running on your server. Sign up for an Oracle account (if needed) and download the correct version. Copy it to your server (i.e. ~ - your home directory for this guide). If you have a mac, you can use scp from the command line, on Windows, you can use WinSCP or similar program. For example to upload from my Mac I use the following scp command (after downloading from Oracle):

```
$ scp -i biorloginkey.pem ~/Downloads/jdk-7u45-linux-x64.tar.gz
ubuntu@10.148.2.10:~/jdk-7u45-linux-x64.tar.gz
```

Yours will differ because your cloud server will be on a different IP and you will have a different credential file (the .pem).

This will place a copy of Java in my home directory (~ = /home/ubuntu) Extract it.

```
ubuntu@biorinstall2:~$ pwd
/home/ubuntu
```

```
ubuntu@biorinstall2:~$ tar -zxvf jdk-7u45-linux-x64.tar.gz
```

Then you will need to put it in your path:

```
ubuntu@biorinstall2:~$ ls
bior_2.2.0  bior_2.2.0.tar.gz  jdk1.7.0_45  jdk-7u45-linux-x64.tar.gz
ubuntu@biorinstall2:~$ JAVA_HOME=/home/ubuntu/jdk1.7.0_45
ubuntu@biorinstall2:~$ PATH=$PATH:$HOME/bin:$JAVA_HOME/bin
ubuntu@biorinstall2:~$ export $JAVA_HOME
-bash: export: `/home/ubuntu/jdk1.7.0_45': not a valid identifier
ubuntu@biorinstall2:~$ export JAVA_HOME
ubuntu@biorinstall2:~$ export PATH
ubuntu@biorinstall2:~$ javac
(usage information should be here, if not it is not correctly installed)
```

I put these commands in ~/.bash\_profile so that they are in place next time I log in.

### Toolkit Installation:

1) Download the toolkit: (e.g.)

```
$ wget https://s3-us-west-2.amazonaws.com/mayo-bic-tools/bior/bior\_2.2.0.tar.gz
```

2) Unzip the bior\_version zip file you downloaded. (unzip bior\_version.zip -d target directory) e.g.:

```
$ tar -xzf bior_2.2.0.tar.gz
```

3) Make sure all your files in bior\_pipeline project are executable.

```
chmod -R ugo+x bior_version directory e.g.
```

```
$ cd bior_2.2.0/
```

```
$ chmod +x bin/
```

4) Now you need to setup the environment variables and add to the path.

```
$ export BIOR_LITE_HOME=YOUR_BIOR_FOLDER
```

```
$ export PATH=$BIOR_LITE_HOME/bin:$PATH
```

There is a quick script that comes with BioR that can help setupEnv.sh. Just source the file:

```
$ source setupEnv.sh
```

You will need to setup your paths each time you login so it might make sense to add this command into your .bash\_profile/.bashrc.

5) Now try bior\_ and press tab key twice on terminal. Now you should see all bior commands displayed. If they are not being displayed, look inside the setupEnv.sh and change the paths so that they work with your environment. (BioR is using BASH). Change it as needed or ask a system administrator for help. Make sure you can type bior\_(tab tab) and get all of the commands back before moving on to the next step.

6) Verify that it is installed correctly by typing bior\_pretty\_print -h. You should see a help message, if you see an error like "java: not found" then you need to install java correctly.

Now you have successfully installed the toolkit. The quickstart guide is a good place to go to check if your toolkit is functioning properly and to run some biologically motivated examples contributed from our bioinformaticians (they even use versions of these in production!).

One of the hardest to set up commands is `bior_annotate`. `Bior_annotate` is a kitchen sink command and requires the catalogs, command line tools and all dependencies be installed properly on your system. The next three sections will go over how to install all of the catalogs it needs, and how to install SNPEFF and VEP. For now, it is important to highlight the `bior.properties` file in `$BIOR_LITE_HOME/conf`. For example, mine is here:

```
$ pwd
/home/ubuntu/bior_2.2.0/conf
ubuntu@biorinstall2:~/bior_2.2.0/conf$ ls
allCatalogs.columns.properties allCatalogs.columns.tsv bior.properties cli log4j.properties tools
```

Edit the configuration file so that all tools, catalogs, and paths are consistent with your install locations (see the next section).

## Catalogs Installation:

1) use `wget` to get all of the BioR catalogs and place them in `/data`

I used the following `wget` commands from inside the `/data` directory:

```
wget https://s3-us-west-2.amazonaws.com/mayo-bic-tools/bior/Catalogs/Bior/dbSNP.137.GRCh37.tar.gz &
wget
https://s3-us-west-2.amazonaws.com/mayo-bic-tools/bior/Catalogs/Bior/1000_genomes.phase1_release_v3_20
101123.GRCh37.tar.gz
wget
https://s3-us-west-2.amazonaws.com/mayo-bic-tools/bior/Catalogs/Bior/hapmap.2010_08_phaseII%2BIII.GRCh
37.tar.gz
wget https://s3-us-west-2.amazonaws.com/mayo-bic-tools/bior/Catalogs/Bior/ucsc.hg19.GRCh37.tar.gz
wget https://s3-us-west-2.amazonaws.com/mayo-bic-tools/bior/Catalogs/Bior/ESP.build37.GRCh37.tar.gz
wget https://s3-us-west-2.amazonaws.com/mayo-bic-tools/bior/Catalogs/Bior/cosmic.v63.GRCh37.tar.gz
wget
https://s3-us-west-2.amazonaws.com/mayo-bic-tools/bior/Catalogs/Bior/hgnc.2012_08_12.GRCh37.tar.gz
wget https://s3-us-west-2.amazonaws.com/mayo-bic-tools/bior/Catalogs/Bior/BGI.hg19.GRCh37.tar.gz
wget
https://s3-us-west-2.amazonaws.com/mayo-bic-tools/bior/Catalogs/Bior/NCBIGene.2013_04_08.GRCh37_p10.ta
r.gz
wget
https://s3-us-west-2.amazonaws.com/mayo-bic-tools/bior/Catalogs/Bior/omim.2013_02_27.GRCh37.tar.gz
wget
https://s3-us-west-2.amazonaws.com/mayo-bic-tools/bior/Catalogs/Bior/mirbase.release19.GRCh37_p5.tar.g
z
wget https://s3-us-west-2.amazonaws.com/mayo-bic-tools/bior/Catalogs/User/drugbank3.tar.gz
wget https://s3-us-west-2.amazonaws.com/mayo-bic-tools/bior/Catalogs/User/pharmgkb_2013_07_03.tar.gz
wget https://s3-us-west-2.amazonaws.com/mayo-bic-tools/bior/Catalogs/User/ttd_4.3.02.tar.gz
wget https://s3-us-west-2.amazonaws.com/mayo-bic-tools/bior/Catalogs/User/uniprot_2013_07_10.tar.gz
```

If you like you could just put this in a script and run it.

2) Then use the following command to uncompress all of the catalogs: `ubuntu@biorinstall2:/data$ find . | grep tar.gz | xargs tar -xzf`

This will extract the downloaded catalogs into a directory. Make sure you extract all catalogs into same target directory. Once they are extracted, you can delete the zipped files.

3) Now, for `bior_annotate` to work, you will need to set the properties (for all of the rest of BioR, you are good to go).

You will find a file named *bior.properties* under the folder `conf` in your `bior_version` directory (See the section above on installing the toolkit). This is the file where you need to set the tools path and home path of catalogs directory. Tool commands like `bior_vep` and `bior_snpeff` and as well as `bior_annotate` make use of this properties file. My file is here:

```
$ pwd
/home/ubuntu/bior_2.2.0/conf
$ ls
allCatalogs.columns.properties  allCatalogs.columns.tsv  bior.properties  cli
log4j.properties  tools
```

The production version of this file is highly customized to running inside of the RCF. In this guide, The following content can easily replace the contents of production version. The rest of the setup guide will assume that your bior.properties file is set up as follows (note fileBase = /data/ ), the paths you may need to change are in bold:

```
$ cat bior.properties
#####
### NOTE: These keys are defined in BiorProperties.java and must match the keys in the enum there
#####

###SNPEFF =====
SnpEffJar=/data/snpEff_2_0_5d/snpEff.jar
SnpEffConfig=/data/snpEff_2_0_5d/snpEff.config

###VEP =====
BiorVepPerl=/usr/bin/perl
BiorVep=/data/variant_effect_predictor/variant_effect_predictor.pl
BiorVepCache=/data/variant_effect_predictor/cache/

###BIOR_TREAT =====
### AnnotateMaxLinesInFlight:
### NOTE: Min = 2. Default = 10. Max = 50 (could do more, but not recommended)
### WARNING: Do not increase it to much more than 50 or you may encounter a hang state, especially
with a high number of fanouts, as the process buffers will overflow!
AnnotateMaxLinesInFlight=10
fileBase=/data/
genesFile=NCBIGene/GRCh37_p10/genes.tsv.bgz
bgiFile=BGI/hg19/LuCAMP_200exomeFinal.maf_GRCh37.tsv.bgz
espFile=ESP/build37/ESP6500SI_GRCh37.tsv.bgz
hapMapFile=hapmap/2010-08_phaseII+III/allele_freqs_GRCh37.tsv.bgz
dbsnpFile=dbSNP/137/00-All_GRCh37.tsv.bgz
dbsnpClinvarFile=dbSNP/137/clinvar_20130226_GRCh37.tsv.bgz
cosmicFile=cosmic/v63/CosmicCompleteExport_GRCh37.tsv.bgz
kGenomeFile=1000_genomes/20110521/ALL.wgs.phase1_release_v3.20101123.snps_indels_sv.sites_GRCh37.tsv.b
gz
blacklistedFile=ucsc/hg19/wgEncodeDacMapabilityConsensusExcludable_GRCh37.tsv.bgz
repeatFile=ucsc/hg19/rmsk_GRCh37.tsv.bgz
regulationFile=ucsc/hg19/oreganno_GRCh37.tsv.bgz
uniqueFile=ucsc/hg19/wgEncodeDukeMapabilityRegionsExcludable_GRCh37.tsv.bgz
tssFile=ucsc/hg19/switchDbTss_GRCh37.tsv.bgz
tfbsFile=ucsc/hg19/tfbsConsSites_GRCh37.tsv.bgz
enhancerFile=ucsc/hg19/vistaEnhancers_GRCh37.tsv.bgz
### conservationFile=ucsc/hg19/phastConsElements46wayPrimates_GRCh37.tsv.bgz
conservationFile=ucsc/hg19/phastConsElements46way_GRCh37.tsv.bgz
hgncFile=hgnc/2012_08_12/hgnc_GRCh37.tsv.bgz
hgncIndexFile=hgnc/2012_08_12/index/hgnc_GRCh37.Entrez_Gene_ID.idx.h2.db
hgncEnsemblGeneIndexFile=hgnc/2012_08_12/index/hgnc_GRCh37.Ensembl_Gene_ID.idx.h2.db
omimFile=omim/2013_02_27/genemap_GRCh37.tsv.bgz
omimIndexFile=omim/2013_02_27/index/genemap_GRCh37.MIM_Number.idx.h2.db
mirBaseFile=mirbase/release19/hsa_GRCh37.p5.tsv.bgz
```

Now in the file you need to set fileBase="catalogs directory" value to your catalogs directory.

Some users notice a problem with Sage not being able to make a connection, if you get this error, make sure the SAGE\_ENVIRONMENT is set as follows: SAGE\_ENVIRONMENT=null, (or =prod) in the Global.properties file (found in: /home/ubuntu/bior\_2.2.0/conf/cli on the example system, or

\$BIOR\_LITE\_HOME/conf/cli)

Example : fileBase=/data/

Next step is tools installation.

## Dependant Tools Installation and Setup

We have integrated two tools SNPEff and Variant Effect Predictor (VEP) into our toolkit.

### Variant Effect Predictor (VEP):

The Version of VEP we support is 2.7.

[http://useast.ensembl.org/info/docs/tools/vep/script/vep\\_download.html#versions](http://useast.ensembl.org/info/docs/tools/vep/script/vep_download.html#versions)

You can follow the installation instructions in the above page. Here is how we install it step by step:

1) download it from the website and scp it over to your Linux box (note: this is using an ssh key "biorloginkey.pem" that corresponds to the account on the cloud server. Your scp command may not require it, or could use username/password instead):

```
scp -i biorloginkey.pem ~/Downloads/variant_effect_predictor.tar.gz  
ubuntu@10.148.2.10:/data/variant_effect_predictor.tar.gz
```

2) Extract

```
$ tar -xzf variant_effect_predictor.tar.gz
```

3) Make sure you have Perl version 14 on your computer:

```
$ perl -v
```

4) If the perl is the correct version, and you have admin rights, you can now install the Perl libraries needed by VEP:

```
$ sudo apt-get install libwww-perl  
$ sudo perl -MCPAN -e'force install "LWP::Simple"  
$ sudo apt-get install libdbi-perl  
$ sudo apt-get install libdbd-mysql-perl
```

You can also download the Perl libraries separately and point your PERL5LIB environment variable to them. For more information, see:

<http://linuxgazette.net/139/okopnik.html>

5) Then run the VEP installer:

```
perl INSTALL.pl [options]
```

6) Make sure to download the needed cache files or correct the install (in my case the download script failed)

You may follow instructions at [http://www.ensembl.org/info/docs/api/api\\_installation.html](http://www.ensembl.org/info/docs/api/api_installation.html) which provides alternate options for the API installation. Example download of the cache files.

```
$ nohup wget  
ftp://ftp.ensembl.org/pub/release-69/variation/VEP/homo_sapiens_vep_69.tar.gz &
```



```
$ nohup wget
ftp://ftp.ensembl.org/pub/release-69/variation/VEP/homo_sapiens_vep_69_sift_poly
phen.tar.gz &
```

7) unzip the alignment files to a directory called “cache” inside the same directory as VEP.

8) Change the bior.properties file to point at the version of vep that you just installed.

```
###VEP =====
BiorVepPerl=/usr/bin/perl
BiorVep=/data/variant_effect_predictor/variant_effect_predictor.pl
BiorVepCache=/data/variant_effect_predictor/cache/
```

9) Test that VEP works stand-alone on a VCF file to ensure it is installed correctly. Here is an example command:

```
perl /data/variant_effect_predictor/variant_effect_predictor.pl -i example.vcf
-o STDOUT -dir /data/variant_effect_predictor/cache/ -vcf -polyphen b -sift b
--offline
```

10) Test that VEP works inside the BioR wrapper:

```
cat example.vcf | bior_vep > vepAnnotated.tjson
```

## SNPEff:

Currently we support SNPEff version 2.0.5d. This was recommended by GATK for worst pick logic.

Installation files and instructions can be found at

<http://snpeff.sourceforge.net/download.html>

If you are using Linux or Mac you can just use wget command to download the files below.

[http://sourceforge.net/projects/snpeff/files/snpEff\\_v2\\_0\\_5d\\_core.zip](http://sourceforge.net/projects/snpeff/files/snpEff_v2_0_5d_core.zip)

Database you need to download is at

[http://sourceforge.net/projects/snpeff/files/databases/v2\\_0\\_5/snpEff\\_v2\\_0\\_5\\_GRCh37.64.zip](http://sourceforge.net/projects/snpeff/files/databases/v2_0_5/snpEff_v2_0_5_GRCh37.64.zip)

Make sure to change SNPEFF config file snpEff.config to include the path to the database you downloaded.

1) Make sure you have unzip installed so you can extract the zip file:

```
$ sudo apt-get install unzip
```

2) Extract SNPEFF

```
$ cd /data
$ wget http://sourceforge.net/projects/snpeff/files/snpEff\_v2\_0\_5d\_core.zip
$ unzip snpEff_v2_0_5d_core.zip
```

3) By default, SNPEFF expects that you place the data in a directory called ‘data’ residing in the same directory as snpEff.jar (SNPEFF\_HOME). Extract the data for SNPEFF to SNPEFF\_HOME/data.

```
$ cd snpEff_2_0_5d
$ wget
http://sourceforge.net/projects/snpeff/files/databases/v2\_0\_5/snpEff\_v2\_0\_5\_GRCh37.64.zip
$ unzip snpEff_v2_0_5_GRCh37.64.zip
###(This will create the /data/snpEff_2_0_5d/data/ directory)
```

4) After you have installed SNPEff, set the paths in bior.properties file located in conf folder under your bior\_pipeline directory.

Example:

```
###SNPEFF =====  
SnpEffJar=/data/snpEff_2_0_5d/snpEff.jar  
SnpEffConfig=/data/snpEff_2_0_5d/snpEff.config
```

5) check that SNPEFF works as a stand alone tool. This will take at least a minute to load the database before it starts processing vcf lines.

```
cat example.vcf | java -Xmx4g -jar /data/snpEff_2_0_5d/snpEff.jar eff -c  
/data/snpEff_2_0_5d/snpEff.config -v -o vcf -noLog -noStats GRCh37.64
```

6) Run the BioR wrapper for SNPEFF:

```
cat example.vcf | bior_snpeff > annotated.tjson
```

## Installing BioR Tools from Source

Source installation requires that you have both Java 1.7 and Maven installed and on your path. It also requires that you have access to the Mayo NEXUS servers or you place several libraries in your ~/.m2 directory.

If you have troubles installing BioR or compiling it, please contact the BioR Team (dlrstitbiorall@mayo.edu) so we can update the documentation and make the process easier.

## Java Heap Size

On some machines, the default JVM size is 2GB. This is very large for BioR. By default the BioR toolkit is capped at 128M. To change this setting, change the Maven bior\_pipeline/pom.xml (e.g. <jvmOpts>-Xmx128m</jvmOpts>).

## 2. Overview

### Introduction

BioR uses a [Pipe-And-Filter](#) architecture. Data to be annotated by BioR is streamed through a pipeline, a sequence of one or more pipes. Pipes is based on Flow Based Programming by J.P. Morrison. [DataFlow-Article](#), [Flow-Based-Programing](#).

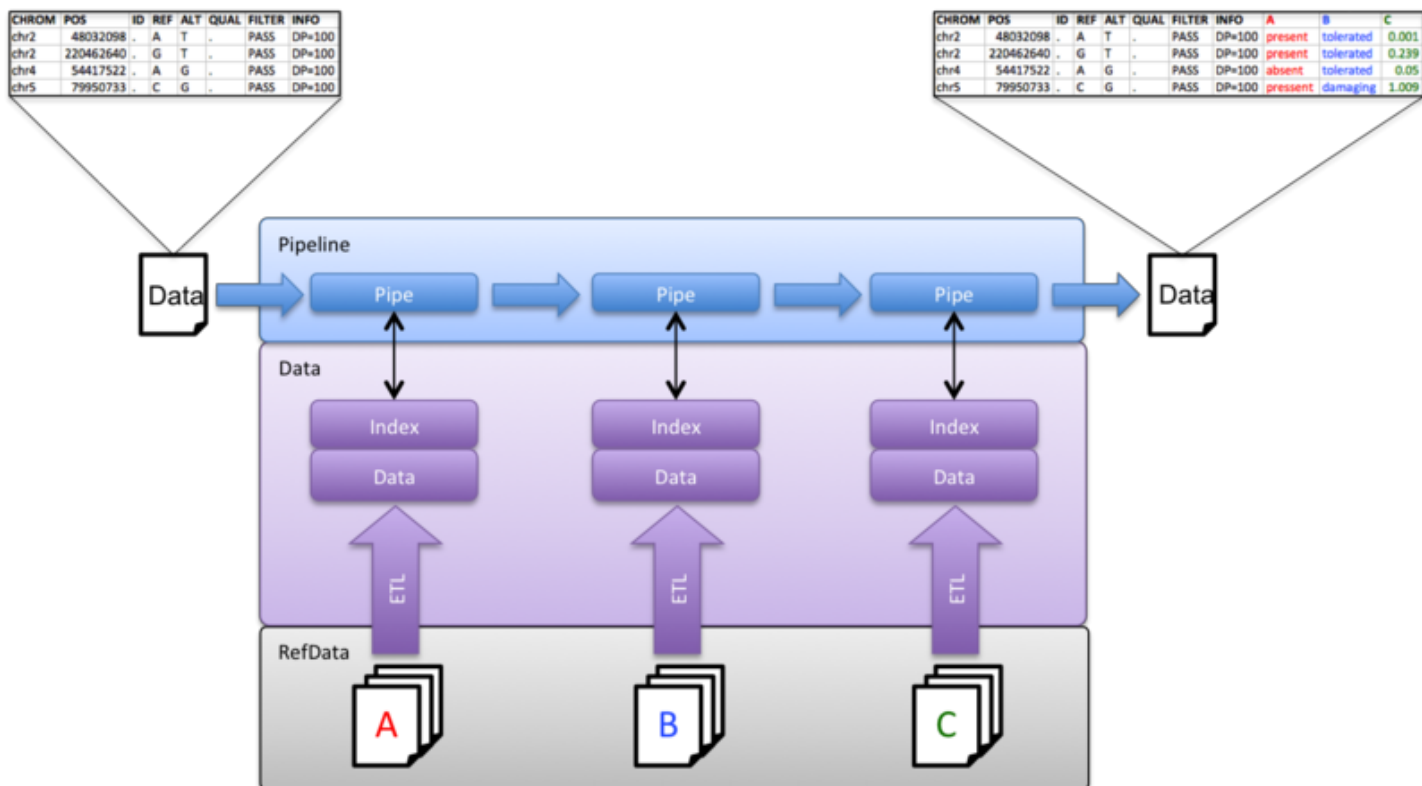
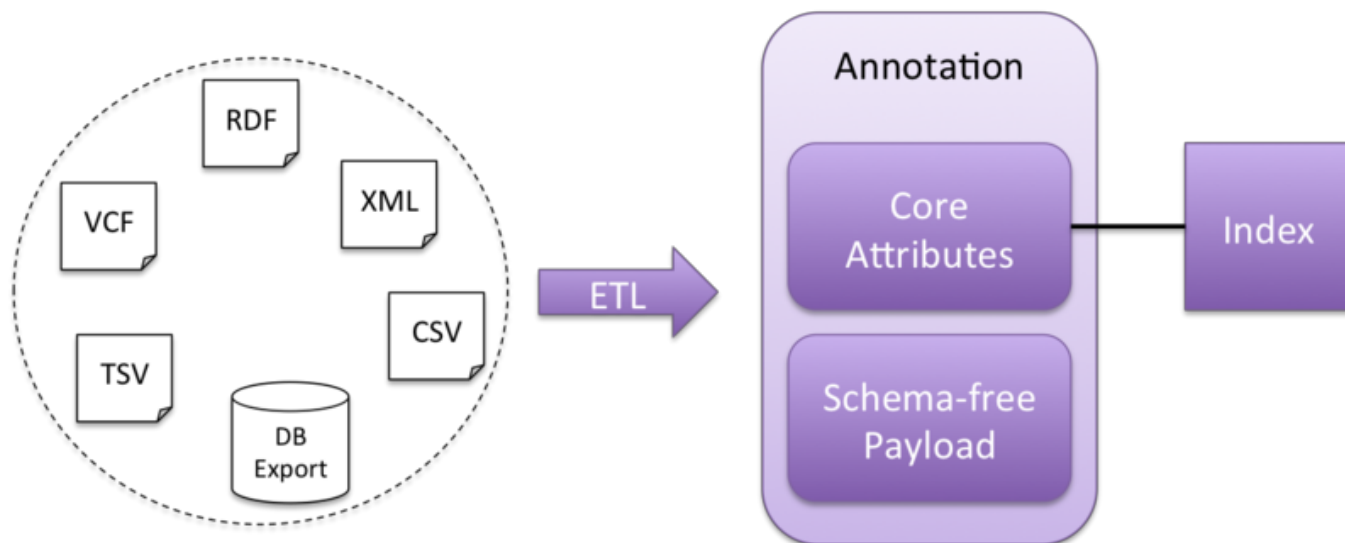


Figure 1: BioRTools works by adding annotation to the right on the original file.

BioR leverages UNIX pipes to flow data from program to program. As BioR programs work on the data, they place annotation to the right (the red, blue and green columns in Figure 1).

## Data Modeling

BioR has adopted a lightweight approach to modeling annotation data. Only **core** annotation fields are modeled to enable supported search capabilities (e.g. coordinate search, accession ID search). Anything not classified as **core** is modeled into a "schema-free" data structure.



## BioR Catalog Shortcut

BioR commands commonly use long paths to files. One of the first things you will want to do when

using BioR is to make an alias to the location of the BioR catalogs. For example if the BioR catalogs are located in `$bior`

Then, on bash, execute the following command at the command line:

```
$ export bior=/data/path/
```

You may want to put this command in your `.bashrc` or `.bash_profile` so that the `$bior` environment variable shows up next time you log in.

## Finding out what is in a Catalog

Each data source is 'published' into a BioR catalog file for use by the BioR scripts. A Catalog is a collection of files (both data and indexes) that is understood by the BioR Pipes infrastructure. BioR's reference data consists of the raw files downloaded/updated and made available to BioR users. These files ARE NOT catalogs. Catalogs are transformed into the BioR standard catalog structure so that pipes can work on the content. BioR catalogs are bgzipped files<sup>1</sup> that contain 4 columns (`_landmark`, `_minBP`, `_maxBP`, and `JSON`). A more comprehensive description of the BioR catalog format is in Chapter 3.

To see what is in a catalog, use the `zcat` command (`gzcat` on a mac) followed by the catalog filename, followed by `less`:

```
$ zcat $bior/NCBIGene/GRCh37_p10/genes.tsv.bgz | less
1      10954   11507
{"_type":"gene","_landmark":"1","_strand":"+","_minBP":10954,"_maxBP":11507,"gene":"LOC100506145","te":"Derived by automated computational analysis using gene prediction method: GNOMON. Supporting evidence includes similarity to: 1 Protein","pseudo":"","GeneID":"100506145"}
...
```

Unix `less` is a good-low-memory command to look at data. Type `q` <enter> to quit `less`. Type `man less` at the command line to see how to use the `less` command. You can use up and down arrows to scroll through the data a line at a time or 'f' and 'b' to scroll a page at a time.

## Showing the Commands in BioR Toolkit

All BioR commands start with `bior_` so once BioRTools is installed and on your path you can type `bior_` followed by the tab key (twice) and it will show you all of the current commands in the toolkit:

```
$ bior_
bior_annotate      bior_create_catalog_props  bior_lookup
bior_snpeff        bior_vep                   bior_bed_to_tjson
bior_create_config_for_tab_to_tjson  bior_overlap               bior_tab_to_tjson
bior_compress      bior_drill                  bior_pretty_print
bior_tjson_to_vcf   bior_create_catalog         bior_index_catalog
bior_same_variant   bior_vcf_to_tjson
```

---

<sup>1</sup> <http://samtools.sourceforge.net/tabix.shtml>

Table 1 has a more complete description of these commands. Commands in the toolkit operate on tab delimited data with a VCF style header (starting with “#”). Commands in the toolkit insert additional annotation to the right. Raw annotation is obtained by comparing JSON objects in columns to JSON objects in catalogs. Table 1.0 shows the format of columns <in,out> of each BioR function. For example bior\_vcf\_to\_tjson takes as an input VCF columns (and the header) and outputs VCF + JSON in the last column.

Command	Input, Output	Description
<b>Transform Functions</b>		
bior_overlap	TJSON, TJSON	Extract annotations from a catalog based on genomic location overlap. The overlap is computed from the Start and End genomics position of a variant.
bior_same_variant	TJSON, TJSON	Extract annotations from a catalog based on variant position, reference and alternate allele definition.
bior_lookup	TJSON, TJSON	Extract annotations from a catalog based on matching values of an identifier.
bior_snpeff	TJSON, TJSON	Use SNPEff <sup>1</sup> to annotate variants. Chromosome ID, Start and Stop genomics position, reference and alternate allele of the variant is required .
bior_vep	TJSON, TJSON	Use VEP <sup>2</sup> to annotate variants. Chromosome ID, Start and Stop genomics position, reference and alternate allele of the variant is required.
bior_drill	TJSON, TJSON	Extract an element from nested JSON string.
bior_compress	TJSON, TJSON	Compress entries from provided set of identifiers into a single entry with each value separated by a delimiter.
<b>Utility Functions</b>		
bior_index_catalog	identifier, index	Index the specified identifier in a catalog. Indices a stored in a separate index file.
bior_create_catalog	TJSON,	Convert a text tabulated file into a catalog.

	catalog	Chromosome ID, Start and End genomics position fields have to be explicitly named.
bior_create_catalog_props	catalog, property	Create property files from the metadata extracted from a catalog. Property files are needed for proper metadata handling.
bior_create_config_for_tab_to_tjson	TSV, config	Create a configuration file that describes column description. This file is needed when uploading a tab delimited file.
<b>Input/Output Functions</b>		
bior_vcf_to_tjson	VCF, TJSON	Load a VCF file and convert to TJSON format.
bior_tjson_to_vcf	TJSON, VCF	Convert TJSON to VCF format for file output.
bior_bed_to_tjson	BED, TJSON	Load a BED file and convert to TJSON format.
bior_tab_to_tjson	TSV, TJSON	Load a tab-delimited file and convert to TJSON format.
bior_pretty_print	TJSON, STDOUT	Convert TJSON in a readable format for screen or file output.
<b>Miscellaneous Functions</b>		
bior_annotate	VCF, TJSON	Append to the VCF 'info' field a set of commonly used annotations.

Table 1: List of commands available in the BioR Toolkit. Detailed description and example is displayed when executing the command with the -h flag.

<sup>1</sup>Cingolani, P. et al. (2012) A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff: SNPs in the genome of *Drosophila melanogaster* strain w1118; iso-2; iso-3. Fly (Austin). 6(2) :p. 80-92.

<sup>2</sup>McLaren W et al. (2010) Deriving the consequences of genomic variants with the Ensembl API and SNP Effect Predictor. BMC Bioinformatics 26(16):2069-70

Most every one of these commands supports the -h (help) flag to get information about how to use the command. To get help on bior\_vcf\_to\_tjson type:

```
$ bior_vcf_to_tjson -h

NAME
bior_vcf_to_tjson -- converts VCF data into JSON as an additional column

SYNOPSIS
bior_vcf_to_tjson [--log] [--help]
```

...

Several of the above functions use ‘Golden Identifiers’ to match records across catalogs. Table 2 shows the current golden identifiers used in the codebase and what function(s) use them.

‘Golden Identifier’	Functions	Definition
landmark	<code>bior_overlap,</code> <code>bior same variant</code>	Chromosome, or sequence ID that the interval is located on
minBP	<code>bior_overlap,</code> <code>bior same variant</code>	Minimum 1-based position (e.g. NCBI coordinates) on the landmark sequence
maxBP	<code>bior_overlap,</code> <code>bior same variant</code>	Maximum 1-based position on the landmark sequence
<code>_refAllele</code>	<code>bior same variant</code>	REF as in VCF standard
<code>_altAlleles</code>	<code>bior same variant</code>	ALT as in VCF standard

## Pretty Print

Data in the 4<sup>th</sup> column of a catalog is stored as JSON. JSON can be deeply nested and hard to read if it is all smashed into one line. BioR has a command `bior_pretty_print` that can make reading JSON text easier. Take the earlier example and replace `less` with `bior_pretty_print`:

```
$ zcat $bior/NCBIGene/GRCh37_p10/genes.tsv.bgz | bior_pretty_print
#  COLUMN NAME  COLUMN VALUE
-  -
1  UNKNOWN_1    1
2  #UNKNOWN_2   10954
3  #UNKNOWN_3   11507
4  #UNKNOWN_4   {
    "_type": "gene",
    "_landmark": "1",
    "_strand": "+",
    "_minBP": 10954,
    "_maxBP": 11507,
    "gene": "LOC100506145",
    "note": "Derived by automated computational analysis using gene prediction method:
GNOMON. Supporting evidence includes similarity to: 1 Protein",
    "pseudo": "",
    "GeneID": "100506145"
  }
$
```

Use `-r` to specify the row to pretty print. This is very useful when handling sparse data, where the values for columns you are interested in do not appear on every line. In JSON if there is no value for a given key, the key is not shown (instead of reporting NULL), so you may need to hunt around in the dataset a bit to find keys of interest.

## Get all Variants in a Gene

Lets do something useful -- say we wanted all genetic variants in VCF format that overlap the BRCA1 gene from dbSNP. This section will illustrate how to use BioR to rapidly build a program that does just that. BioR is executed at the Linux/UNIX command line, so any command that is available at the command line can be used with BioR (grep, cut, sed, awk, perl, ...). Lets start with the echo command to find BRCA1 in the gene catalog.

```
$ echo "BRCA1" | bior_lookup -p gene -d $bior/NCBIGene/GRCh37_p10/genes.tsv.bgz | bior_pretty_print
#  COLUMN NAME  COLUMN VALUE
-  -
1  UNKNOWN_1    BRCA1
2  LookupPipe   {
    "_type": "gene",
    "_landmark": "17",
    "_strand": "-",
    "_minBP": 41196312,
    "_maxBP": 41277500,
    "gene": "BRCA1",
    "gene_synonym": "BRCAI; BRCC1; BROVCA1; IRIS; PNCA4; PPP1R53; PSCP; RNF53",
    "note": "breast cancer 1, early onset; Derived by automated computational analysis
using gene prediction method: BestRefseq.",
    "GeneID": "672",
    "HGNC": "1100",
    "HPRD": "00218",
    "MIM": "113705"
  }
```

The UNIX pipe (`|`) allows you to stream the output of one command to the next. In this example, echo prints BRCA1 to the screen. `bior_lookup` uses this ID to find the entry in the gene catalog with the key gene and value 'BRCA1'. Now we have the genomic coordinates for BRCA1. Lets use these positions to find all catalog entries in dbSNP that are between 41196312 and 41277500 on chromosome 17.

```
$ echo "BRCA1" | bior_lookup -p gene -d $bior/NCBIGene/GRCh37_p10/genes.tsv.bgz | bior_overlap -d
$bior/dbSNP/137/00-All_GRCh37.tsv.bgz | bior_pretty_print
#  COLUMN NAME  COLUMN VALUE
-  -
```



```

1 UNKNOWN_1 BRCA1
2 LookupPipe {
    "_type": "gene",
    "_landmark": "17",
    "_strand": "-",
    "_minBP": 41196312,
    "_maxBP": 41277500,
    "gene": "BRCA1",
    "gene_synonym": "BRCAI; BRCC1; BROVCA1; IRIS; PNCA4; PPP1R53; PSCP; RNF53",
    "note": "breast cancer 1, early onset; Derived by automated computational analysis
using gene prediction method: BestRefseq.",
    "GeneID": "672",
    "HGNC": "1100",
    "HPRD": "00218",
    "MIM": "113705"
}
3 OverlapPipe {
    "CHROM": "17",
    "POS": "41196363",
    "ID": "rs8176320",
    "REF": "C",
    "ALT": "T",
    "QUAL": ".",
    "FILTER": ".",
    "INFO": {
        "RSPOS": 41196363,
        "RV": true,
        "GMAF": 0.0050,
        "dbSNPBuildID": 117,
        "SSR": 0,
        "SAO": 0,
        "VP": "0500000800201040517000100",
        "GENEINFO": "BRCA1:672",
        "WGT": 1,
        "VC": "SNV",
        "REF": true,
        "U3": true,
        "VLD": true,
        "HD": true,
        "GNO": true,
        "KGPhase1": true,
        "KGPROD": true,
        "OTHERKG": true,
        "PH3": true
    },
    "_id": "rs8176320",
    "_type": "variant",
    "_landmark": "17",
    "_refAllele": "C",

```

```

    "_altAlleles": [
      "T"
    ],
    "_minBP": 41196363,
    "_maxBP": 41196363
  }
$

```

This command shows the first match in dbSNP that overlaps the BRCA1 gene according to the NCBI annotation. The version of dbSNP used to publish the catalog was a VCF file, therefore many fields from the VCF standard are represented in the JSON. A combination of the UNIX `cut` command and `bior_drill` can quickly extract a VCF file. When trying this example, decompose the commands and use them one at a time to understand what each command is doing.

```

$ echo "BRCA1" | bior_lookup -p gene -d $bior/NCBIGene/GRCh37_p10/genes.tsv.bgz | bior_overlap -d
$bior/dbSNP/137/00-All_GRCh37.tsv.bgz | bior_drill -p CHROM -p POS | cut -f 1,3,4 | head -10

##BIOR=<ID="bior.gene37p10",Operation="bior_lookup",DataType="JSON",ShortUniqueName="gene37p10",Sou
e="NCBIGene",Description="NCBI's Gene Annotation directly from the gbs
file",Version="37p10",Build="GRCh37.p10",Path="/data5/bsi/catalogs/bior/v1/NCBIGene/GRCh37_p10/gene
tsv.bgz">
##BIOR=<ID="bior.dbSNP137",Operation="bior_overlap",DataType="JSON",ShortUniqueName="dbSNP137",Sour
="dbSNP",Description="NCBI's dbSNP Variant
Database",Version="137",Build="GRCh37.p5",Path="/data5/bsi/catalogs/bior/v1/dbSNP/137/00-All_GRCh37
sv.bgz">
##BIOR=<ID="bior.dbSNP137.CHROM",Operation="bior_drill",Field="CHROM",DataType="String",Number="1",F
eldDescription="Chromosome. (VCF
field)",ShortUniqueName="dbSNP137",Source="dbSNP",Description="NCBI's dbSNP Variant
Database",Version="137",Build="GRCh37.p5",Path="/data5/bsi/catalogs/bior/v1/dbSNP/137/00-All_GRCh37
sv.bgz">
##BIOR=<ID="bior.dbSNP137.POS",Operation="bior_drill",Field="POS",DataType="Integer",Number="1",Fie
Description="The reference position, with the 1st base having position 1. (VCF
field)",ShortUniqueName="dbSNP137",Source="dbSNP",Description="NCBI's dbSNP Variant
Database",Version="137",Build="GRCh37.p5",Path="/data5/bsi/catalogs/bior/v1/dbSNP/137/00-All_GRCh37
sv.bgz">
#UNKNOWN_1      bior.dbSNP137.CHROM      bior.dbSNP137.POS
BRCA1      17      41196363
BRCA1      17      41196368
BRCA1      17      41196372
BRCA1      17      41196403
BRCA1      17      41196408

```

The result: a simple VCF-like file constructed for all variants in the BRCA1 gene! There are a few small fixes that will need to be made to make it truly VCF-compliant, and this quickstart glosses over many features such as the metadata and headers. These and many other issues will be covered in more detail in the following sections.

### 3. BioR Catalogs

#### The BioR Catalog Format

BioR enables users to rapidly transform tabular, hierarchical (e.g. XML) relational, and flat files into catalogs that can be indexed and searched. Catalogs are read-only snapshots of annotation data. In production, we snapshot data sets from outside groups and run an automated ‘publishing’ process that keeps all of the BioR catalogs up to date with reference data sources. Data in catalogs is organized as a BED-JSON hybrid (a subset of TJSON, or tab-delimited JSON). Columns 1-3 are identical to the required fields in BED files<sup>2,3</sup> and thus allow many existing tools such as Tabix to work directly on BioR catalogs. Column 4 is a JSON string encoded object representing the entire contents of the original file. BioRTools depends on *golden identifiers* (identifiers that start with an underscore) to enable search. *Golden identifiers* are semantically-consistent tightly-controlled fields that are used by the toolkit to enable filtering and search (e.g. `_minBP/_maxBP` corresponds to one-based fully-closed genomic min/max base-pairs).

#### Catalog Creation Details

As an illustration, we will take a single gene BRCA1 and show it in the original annotation file and in BioR Catalog structure.

##### ORIGINAL

The gene BRCA1 is shown below from the original Genbank formatted file:  
hs\_ref\_GRCh37.p10\_chr17.gbs.gz:

```
gene          complement(41196312..41277500)
              /gene="BRCA1"
              /gene_synonym="BRCAI; BRCC1; BROVCA1; IRIS; PNCA4;
              PPP1R53; PSCP; RNF53"
              /note="breast cancer 1, early onset; Derived by automated
              computational analysis using gene prediction method:
              BestRefseq."
              /db_xref="GeneID:672"
              /db_xref="HGNC:1100"
              /db_xref="HPRD:00218"
              /db_xref="MIM:113705"
```

##### CATALOG

Below is the corresponding Catalog structure for the final column of gene BRCA1.

```
{
  "gene": "BRCA1",
  "gene_synonym": "BRCAI; BRCC1; BROVCA1; IRIS; PNCA4; PPP1R53; PSCP; RNF53",
  "note": "breast cancer 1, early onset; Derived by automated computational analysis using gene
prediction method: BestRefseq.",
  "GeneID": "672",
  "HGNC": "1100",
  "HPRD": "00218",
  "MIM": "113705",
  "_type": "gene",
  "_landmark": "17",
```

```

    "_strand": "-",
    "_minBP": 41196312,
    "_maxBP": 41277500
  }

```

The catalog format is simple, easy to read, and can be readily processed by third party JSON libraries. The format is also incredibly flexible, and has allowed us to ingest deeply nested XML structures and complex relational schemas into BioR. Construction of catalogs can be done with whatever programming language the user is familiar with. Once the raw data is formatted, bgzip and tabix can be used to compress and then index the catalog for genomic coordinate-based queries.

## Catalogs Available In BioR

The BioR team has created more than 8,000 catalogs relevant to variant annotation from the following sources.

Data sources currently available in BioR

Datasource	URL	Version
1000Genomes	<a href="http://www.1000genomes.org/category/ftp">http://www.1000genomes.org/category/ftp</a>	20110521
BGI	<a href="http://soap.genomics.org.cn/soapsnp.html">http://soap.genomics.org.cn/soapsnp.html</a>	hg19
COSMIC	<a href="http://cancer.sanger.ac.uk/cancergenome/projects/cosmic/">http://cancer.sanger.ac.uk/cancergenome/projects/cosmic/</a>	V63
dbSNP	<a href="http://www.ncbi.nlm.nih.gov/snp/">http://www.ncbi.nlm.nih.gov/snp/</a>	137
ESP6500	<a href="https://esp.gs.washington.edu/drupal/">https://esp.gs.washington.edu/drupal/</a>	build37
HapMap	<a href="http://hapmap.ncbi.nlm.nih.gov">http://hapmap.ncbi.nlm.nih.gov</a>	2010-08_phaseII+III
HGNC	<a href="http://www.genenames.org">http://www.genenames.org</a>	2012_08_12
miRBase	<a href="http://www.mirbase.org">http://www.mirbase.org</a>	8_12_12
NCBIGene	<a href="http://www.ncbi.nlm.nih.gov/gene">http://www.ncbi.nlm.nih.gov/gene</a>	GRCh37_p10
OMIM	<a href="http://www.omim.org">http://www.omim.org</a>	2013_02_27
PharmGKB	<a href="http://www.pharmgkb.org/downloads/">http://www.pharmgkb.org/downloads/</a>	June 2013
DrugBank	<a href="http://www.drugbank.ca/downloads">http://www.drugbank.ca/downloads</a>	3.0
Therapeutic Target Database	<a href="http://bidd.nus.edu.sg/group/cjttd/TTD_Download.asp">http://bidd.nus.edu.sg/group/cjttd/TTD_Download.asp</a>	4.3.02
UCSC	<a href="http://hgdownload.cse.ucsc.edu/goldenPath/hg19/database/">http://hgdownload.cse.ucsc.edu/goldenPath/hg19/database/</a> (note catalogs were created for each UCSC track)	hg19

Table S3: list of data sources from which BioR catalogs are derived. A description of the catalog is available at <http://bioinformaticstools.mayo.edu>

## Creating a Catalog in Seconds

Many users have data in an arbitrary format (e.g. an Excel file from a paper) or another source of annotation such as BioMART. BioR allows users to integrate additional sources of information into the system as catalogs extremely rapidly. Unlike most other tools, BioR does not require a specific bioinformatics format, all you need to be able to do is convert the files into JSON, and BioR has many utilities to do that for you!

As an example, lets integrate dbSNFP2.1 into BioR (available from here: <https://sites.google.com/site/jpopgen/dbNSFP>).

### First Genes:

```
$ head -n 1 dbNSFP2.1_gene | tr "(" "_" | tr ")" "_" |  
bior_create_config_for_tab_to_tjson > gene.config  
$ vim gene.config (to identify the _landmark golden identifier)  
$ cat dbNSFP2.1_gene | bior_tab_to_tjson -c gene.config > dbNSFP2.1_gene.tjson  
$ bior_create_catalog -c -1 -i dbNSFP2.1_gene.tjson -o dbNSFP2.1_gene
```

### Then Variants:

```
$ cat dbNSFP2.1_variant* | grep -v "^#" > dbNSFP2.1_variant  
$ head -n 1 dbNSFP2.1_variant.chr1|bior_create_config_for_tab_to_tjson >  
variant.config  
$ vim variant.config (to columns for _landmark, _minBP, _maxBP, _refAllele, and  
_altAllele)  
$ cat dbNSFP2.1_gene | bior_tab_to_tjson -c gene.config > dbNSFP2.1_gene.tjson  
$ bior_create_catalog -c -1 variant.tjson -o dbNSFP2.1_variant
```

It is really that simple, now dbNSFP is integrated into BioR! To use it, make sure to index as needed using `bior_index_catalog` command.

## 4. Examples Matching Genomic Features

### Positional Matches Using Tabix

BioR uses the same technology for compression (BGZIP) and coordinate based indexing as Tabix<sup>2</sup>. This means that coordinate-based queries can use the traditional Tabix commands. For example, to show all genes in a BioR catalog on Chromosome 17 in the range 41196312 - 41277500:

```
$ which tabix  
/usr/bin/tabix  
  
$ which bgzip  
/usr/bin/bgzip
```

---

<sup>2</sup> <http://bioinformatics.oxfordjournals.org/content/27/5/718.abstract>

```
$ tabix $bior/NCBIGene/GRCh37_p10/genes.tsv.bgz 17:41196312-41277500
17 41196312 41277500
{"_type":"gene","_landmark":"17","_strand":"-","_minBP":41196312,"_maxBP":41277500,"gene":"BRCA1","
ne_synonym":"BRCA1; BRCC1; BROVCA1; IRIS; PNCA4; PPP1R53; PSCP; RNF53","note":"breast cancer 1, ear
onset; Derived by automated computational analysis using gene prediction method:
BestRefseq.","GeneID":"672","HGNC":"1100","HPRD":"00218","MIM":"113705"}
174123127841231833{"_type":"gene","_landmark":"17","_strand":"+","_minBP":41231278,"_maxBP":4123183
"gene":"RPL21P4","gene_synonym":"RPL21_58_1548","note":"ribosomal protein L21 pseudogene 4; Derived
by automated computational analysis using gene prediction method: Curated
Genomic.","pseudo":"","GeneID":"140660","HGNC":"17959"}
```

On the Mayo RCF servers, tabix is located at: /projects/bsi/bictools/apps/alignment/tabix/0.2.5/tabix. You may need to type something like /usr/bin/tabix instead of just tabix if it is not in your path (/usr/bin is usually in your path). To put it in your path edit your \$PATH environment variable. In bash this is done by typing export PATH=\$PATH:/usr/bin

## Annotating Variants with Genes that Overlap

A common and simple use of BioR is to ask what genes overlap variants of interest. NCBI Generates an annotation of genes that they store here: [ftp.ncbi.nih.gov/genomes/Homo\\_sapiens](ftp.ncbi.nih.gov/genomes/Homo_sapiens)

This set of files is one of the authoritative sources for storing both the IDs for genes and the genomic coordinates. Unfortunately the gbs file is hard to use without the use of libraries. BioR allows you to do many quick and dirty analyses based on the position of genes. The following example assumes a VCF-like file with only 8 columns e.g.:

```
$ head example.vcf
##fileformat=VCFv4.0
#CHROM POS ID REF ALT QUAL FILTER INFO
12 1584 8808 rs116645811 G A ...
21 2696 5148 rs1135638 G A ...
21 2696 5172 rs010576 T C ...
21 2696 5205 rs1057885 T C ...
21 2697 6144 rs116331755 A G ...
21 2697 6222 rs7278168 C T ...
21 2697 6237 rs7278284 C T ...
21 2697 8790 rs75377686 T C ...
```

Now, lets annotate these variants based on the genes they overlap:

```
$ cat example.vcf | bior_vcf_to_tjson | bior_overlap -d $bior/NCBIGene/GRCh37_p10/genes.tsv.bgz |
bior_drill -p GeneID -p gene | cut -f 9 --complement > example.vcf.genes
$ head example.vcf.genes
##fileformat=VCFv4.0
#CHROMPOSIDREFALTQUALFILTERINFOGeneIDgene
1215848808rs116645811GA...7399USH2A
2126965148rs1135638GA...54148MRPL39
2126965172rs010576TC...54148MRPL39
```

```
2126965205rs1057885TC...54148MRPL39
2126976144rs116331755AG...54148MRPL39
2126976222rs7278168CT...54148MRPL39
2126976237rs7278284CT...54148MRPL39
2126978790rs75377686TC...54148MRPL39
```

```
$ cat example.vcf | bior_vcf_to_tjson | bior_overlap -d
$ bior/NCBIGene/GRCh37_p10/genes.tsv.bgz | bior_drill -p GeneID -p gene | cut -f 9 --
complement > example.vcf.genes
$ head example.vcf.genes
##fileformat=VCFv4.0
#CHROMPOSIDREFALTQUALFILTERINFOGeneIDgene
1215848808rs116645811GA...7399USH2A
2126965148rs1135638GA...54148MRPL39
2126965172rs010576TC...54148MRPL39
2126965205rs1057885TC...54148MRPL39
2126976144rs116331755AG...54148MRPL39
2126976222rs7278168CT...54148MRPL39
2126976237rs7278284CT...54148MRPL39
2126978790rs75377686TC...54148MRPL39
$
```

Feel free to use `bior_pretty_print` instead of `bior_drill` to explore the data. Try drilling out other columns. In-fact, if anything is unclear, break the command apart and run parts of the command to get a better understanding of what steps are doing (e.g. run `cat`, then `cat | bior_vcf_to_tjson | bior_pretty_print`, then `cat | bior_vcf_to_tjson | bior_overlap | bior_pretty_print`, and so on to understand the transformations done in the pipeline).

This is a simple script based on the above technique to show the genes that contain variants in your VCF file:

```
$ $ head example.vcf
##fileformat=VCFv4.0
#CHROMPOSIDREFALTQUALFILTERINFO
1215848808rs116645811GA...
2126965148rs1135638GA...
2126965172rs010576TC...
2126965205rs1057885TC...
2126976144rs116331755AG...
2126976222rs7278168CT...
2126976237rs7278284CT...
2126978790rs75377686TC...
$
```

In many examples, more than one gene may overlap a variant. By default, BioR will ‘fan-out’ the rows replicating each input row for each result in the result set.

Here is an example of a quick script to look for rsIDs in an entire exome sequencing run (followed by variant calling formatted as VCF) where we annotate the rsID-gene relationships:

```
$ cat /data2/bsi/staff_analysis/m088341/BioR/exome_test/s_P68.variants.final.
vcf | cut -f 3 | grep -v "\." | bior_lookup -p ID -d $bior/dbSNP/137/00-
All_GRCh37.tsv.bgz | grep -v "##" | grep -v "^ID" | bior_overlap -d
$bior/NCBIGene/GRCh37_p10/genes.tsv.bgz | bior_drill -p gene | cut -f 2 --
complement | head
#UNKNOWN_lgene
rs146405013LINC00115
rs3115849LINC00115
rs61768173LINC00115
rs4970461LOC100130417
rs4372192SAMD11
rs6605066SAMD11
rs6672356SAMD11
rs6605067SAMD11
rs6685867NOC2L
```

This is one way to get the variants that overlap more than one gene:



```
$ cat /data2/bsi/staff_analysis/m088341/BioR/exome_test/s_P68.variants.final.vcf | cut -f 3 | grep -v "\." | bior_lookup -p ID -d
SBIOR_CATALOG/dbSNP/137/00-All_GRCh37.tsv.bgz | grep -v "##" | grep -v "^ID" |
bior_overlap -d $bior/NCBIGene/GRCh37_p10/genes.tsv.bgz | bior_drill -p gene
| cut -f 2 --complement | grep -v "#UNKNOWN" | grep -v "\." | cut -f 1 | uniq
-c | grep -v "1 rs"
    2 rs6605067
    2 rs2839
    2 rs262688
    2 rs1043703
    2 rs17692
    2 rs2294532
    2 rs1043683
    2 rs1043681
    2 rs10523
    2 rs649639
...
```

In this case, the variants are sorted, so `uniq` can be used directly, but in other cases, consider the `unix sort` command (right before `uniq`). How many variants overlap at least two genes in this exome sample?

```
$ $ wc -l moreThan1.rsID
3778 moreThan1.rsID
```

## Compressing output to enforce 1-1 semantics

Lets say we want to enforce 1-in/1-out semantics (no duplicated variants), BioR has a utility (`bior_compress`) that can help with that. Here we will start directly with the rare variants. A simple `sed` command replaces the counts and gets us back to rsIDs.

```
$ sed 's/      .* //' < moreThan1.rsID
rs6605067
rs2839
rs262688
rs1043703
rs17692
rs2294532
rs1043683
rs1043681
rs10523
rs649639
...
```

Now we can annotate them in much the same way as before: (or we could modify the above pipeline – probably want to do that when we want to keep all the input data, but this gives us example variants that overlap two genes quickly). Run this example without `bior_compress` to see the default behavior when there is more than one result for a row.

```
$ sed 's/      .* //' < moreThan1.rsID | bior_lookup -p ID -d
SBIOR_CATALOG/dbSNP/137/00-All_GRCh37.tsv.bgz | bior_overlap -d
$ bior/NCBIGene/GRCh37_p10/genes.tsv.bgz | bior_drill -p gene | cut -f 1,3 |
bior_compress 2 | head
#UNKNOWN_lgene
rs6605067SAMD11|NOC2L
rs2839SAMD11|NOC2L
rs262688PRKCZ|LOC100506504
rs1043703THAP3|DNAJC11
rs17692THAP3|DNAJC11
rs2294532THAP3|DNAJC11
rs1043683THAP3|DNAJC11
rs1043681THAP3|DNAJC11
rs10523THAP3|DNAJC11
$
```

## 5. Expanded Genes (Xrefs)

The HUGO/HGNC table has database cross-references for gene ids and names. The `bior_lookup` command allows us to ‘walk’ these cross references. Here is an example:

```
$ bior_vcf_to_tjson < example.vcf | bior_overlap -d
$bior/NCBIGene/GRCh37_p10/genes.tsv.bgz | bior_drill -p GeneID -p gene | cut -f
9 --complement | bior_lookup -d $bior/hgnc/2012_08_12/hgnc_GRCh37.tsv.bgz -p
Approved_Symbol | bior_drill -p Approved_Symbol -p Entrez_Gene_ID -p
Ensembl_Gene_ID -p UniProt_ID
##fileformat=VCFv4.0
#CHROMPOSIDREFALTQUALFILTERINFOGeneIDgeneApproved_SymbolEntrez_Gene_IDEnsembl_Ge
ne_IDUniProt_ID
121584880rs116645811GA...7399USH2AUSH2A7399ENSG000000042781O75445
2126965148rs1135638GA...54148MRPL39MRPL3954148ENSG000000154719Q9NYK5
2126965172rs010576TC...54148MRPL39MRPL3954148ENSG000000154719Q9NYK5
2126965205rs1057885TC...54148MRPL39MRPL3954148ENSG000000154719Q9NYK5
2126976144rs116331755AG...54148MRPL39MRPL3954148ENSG000000154719Q9NYK5
...
```

Lookup requires that the referenced column (last by default change it with the `-c` flag) is an ID that has been indexed in the source catalog. ID based indexes are stored in a directory called 'index' at the same level in the filesystem as the catalog. For example, here are all of the indexes for the HGNC catalog:

## Indexing Catalogs

```
$ ls $bior/hgnc/2012_08_12/index/
hgnc_GRCh37.Approved_Symbol.idx.h2.db  hgnc_GRCh37.Entrez_Gene_ID.idx.h2.db
hgnc_GRCh37.UniProt_ID.idx.h2.db
hgnc_GRCh37.Ensembl_Gene_ID.idx.h2.db  hgnc_GRCh37.HGNC_ID.idx.h2.db
```

On the RCF, the administrators are very restrictive about space, so additional indexes must be placed in user/project space. Stand-alone installs can easily place all indexes in the index directory directly under the directory the catalog is in. BioR allows users to make additional indexes through the `bior_index_catalog` command. The help documentation contains:

```
1) bior_index -d $BIOR_CATALOG/NCBIGene/GRCh37_p10/genes.tsv.bgz -p HGNC
OR
2) bior_index -d $BIOR_CATALOG/NCBIGene/GRCh37_p10/genes.tsv.bgz -p HGNC -i
/data/myindexes/genes.HGNC.idx.h2.db
```

Option 1, used by the BioR team to create indexes, will create the index file in the index folder in the same directory as the catalog (as shown in the example for hgnc above). Option 2, most often used by

BioR end users, creates the index in any directory. When using an index created via the second method, you need to adjust the lookup command appropriately. This will be covered more comprehensively in the section on creating custom catalogs.

To make an index, use `bior_pretty_print` to show the contents of the catalog, and then run the index command.

## Looking Up Information about a Gene

Say we wanted to find "Approved\_Symbol", "Entrez\_Gene\_ID", "Ensembl\_Gene\_ID", "UniProt\_ID", and other common alternative symbols for every gene we have in a list. We can use the BioR lookup command:

First, we don't know the catalog Structure of HGNC, here is a way to look at the structure of a catalog:



```

2 #UNKNOWN_2 0
3 #UNKNOWN_3 0
4 #UNKNOWN_4 {
    "HGNC_ID": "HGNC:5",
    "Approved_Symbol": "A1BG",
    "Approved_Name": "alpha-1-B glycoprotein",
    "Status": "Approved",
    "Locus_Type": "gene with protein product",
    "Locus_Group": "protein-coding gene",
    "Previous_Symbols": [],
    "Previous_Names": [],
    "Synonyms": [],
    "Name_Synonyms": [],
    "Chromosome": "19q",
    "Date_Approved": "1989-06-30",
    "Date_Modified": "2010-07-08",
    "Accession_Numbers": [],
    "Enzyme_IDs": [],
    "Entrez_Gene_ID": "1",
    "Ensembl_Gene_ID": "ENSG00000121410",
    ...
    "Pubmed_IDs": [
        "2591067"
    ],
    "RefSeq_IDs": [
        "NM_130786"
    ],
    "Record_Type": "Standard",
    "Primary_IDs": [],
    "Secondary_IDs": [],
    "CCDS_IDs": [
        "CCDS12976.1"
    ],
    "VEGA_IDs": [],
    "mapped_GDB_ID": "GDB:119638",
    "mapped_Entrez_Gene_ID": "1",
    "mapped_OMIM_ID": "138670",
    "mapped_RefSeq": "NM_130786",
    "UniProt_ID": "P04217",
    "mapped_Ensembl_ID": "ENSG00000121410",
    "UCSC_ID": "uc002qsd.4",
    "mapped_Mammal_Gene_Catalog_ID": "MGC-315223"
}

```

To join the information in this catalog, to the information that we have collected in the gene table, we need to tell bior what field in the HGNC table matches the LAST column in our sample data + annotation. In this case, we will join on approved symbol (note: if you ever get an error with doing a lookup, you may need an index file - look into the bior\_index\_catalog command documentation, using -h for help, or contact the bior team for help – running bior commands ).



```
[m102417@crick4 ~]$ cat mygenes.txt
MRPL39
PANX2
BRCA1
[m102417@crick4 ~]$ cat mygenes.txt | bior_lookup -d
$bior/hgnc/2012_08_12/hgnc_GRCh37.tsv.bgz -p Approved_Symbol
#UNKNOWN_1LookupPipe
MRPL39{"HGNC_ID":"HGNC:14027","Approved_Symbol":"MRPL39","Approved_Name":"
mitochondrial ribosomal protein L39","Status":"Approved","Locus_Type":"gene
with protein product","Locus_Group":"protein-coding gene","Previous_Symbols":
[],"Previous_Names":[],"Synonyms":["RPML5","MRP-L5","MGC104174","PRED66","
PRED22","C21orf92","L39mt","MSTP003","MGC3400","FLJ20451"],"Name_Synonyms":
[],"Chromosome":"21q11.2-q21","Date_Approved":"2001-02-28","Date_Modified":"
2012-09-13","Accession_Numbers":["AB051346"],"Enzyme_IDs":[],"
Entrez_Gene_ID":"54148","Ensembl_Gene_ID":"ENSG00000154719","
Mouse_Genome_Database_ID":"MGI:1351620","Specialist_Database_Links":"<!--,-->
<!--,--> <!--,--> <!--,--> <!--,--> <!--,--> <!--,--> <!--,--> <!--,
--> <a href=\"http://www.sanger.ac.uk/perl/genetics/CGP/cosmic?
action=genes&ln=MRPL39\">COSMIC</a><!--,--> <!--,--> <!--,--> <!--,--> <!--,-->
--> <!--,--> ", "Specialist_Database_IDs":["","","","","","","","","",""],
MRPL39","","","","","","","",""], "Pubmed_IDs":["11543634"], "RefSeq_IDs":
["NM_017446"], "Gene_Family_Tag":"MRPL", "Gene_family_description":"\
Mitochondrial ribosomal proteins / large subunits\","Record_Type":
"Standard", "Primary_IDs":[],"Secondary_IDs":[],"CCDS_IDs":["CCDS13573.1",
"CCDS33522.1"], "VEGA_IDs":["OTTHUMG00000078371"], "mapped_GDB_ID":"GDB:
11503068", "mapped_Entrez_Gene_ID":"54148", "mapped_OMIM_ID":"611845",
"mapped_RefSeq":"NM_017446", "UniProt_ID":"Q9NYK5", "mapped_Ensembl_ID":
"ENSG00000154719", "UCSC_ID":"uc002yln.3", "mapped_Mouse_Genome_Database_ID":
"MGI:1351620")
PANX2...
...
$
```

Now lets extract Entrez\_Gene\_ID, Ensembl\_Gene\_ID, and UniProt\_ID from the catalog:

```
[m102417@crick4 ~]$ cat mygenes.txt | bior_lookup -d
/data5/bsi/catalogs/bior/v1/hgnc/2012_08_12/hgnc_GRCh37.tsv.bgz -p
Approved_Symbol | bior_drill -p Entrez_Gene_ID -p Ensembl_Gene_ID -p
UniProt_ID
#UNKNOWN_1Entrez_Gene_IDEnsembl_Gene_IDUniProt_ID
MRPL3954148ENSG00000154719Q9NYK5
PANX256666ENSG00000073150Q96RD6
BRCA1672ENSG00000012048P38398
[m102417@crick4 ~]$
```

## Example of Walking Cross References

The HGNC table does not contain information about the disease/condition, only the ID in OMIM. Lets say you would like to also find this information for a select set of genes. In this case, we can use two

catalogs, (1) the HGNC catalog and (2) the genemap directly from OMIM. The figure below shows the contents of the genemap catalog currently in BioR:

```
$ zcat $bior/omim/2013_02_27/genemap_GRCh37.tsv.bgz | bior_pretty_print
# COLUMN NAME COLUMN VALUE
- - - - -
1 UNKNOWN_1 .
2 #UNKNOWN_2 .
3 #UNKNOWN_3 .
4 #UNKNOWN_4 {
    "Chromosome.Map_Entry_Number": 1.1,
    "MonthEntered": 9,
    "Day": 11,
    "Year": 95,
    "Cytogenetic_location": "1pter-p36.13",
    "GeneSymbols": "CCV",
    "Gene_Status": "P",
    "Title": "Cataract, congenital, Volkmann type",
    "Title_cont": "",
    "MIM_Number": 115665,
    "Method": "Fd",
    "Comments": "",
    "Disorders": "Cataract, congenital, Volkmann type (2)",
    "Disorders_cont": " "
}
```

In this catalog, "MIM\_Number" represents the OMIM id for the "Disorder" free text field describing the disease. Given a list of genes, if we want the value of the "Disorder" field in OMIM we can cross-walk from the gene list through the HGNC catalog to find the MIM number and then again to genemap catalog to produce a Gene-OMIM\_ID-Disorder file:

```
$ cat mygenes.txt
MRPL39
PANX2
BRCA1
$ cat mygenes.txt | bior_lookup -d $bior/hgnc/2012_08_12/hgnc_GRCh37.tsv.bgz
-p Approved_Symbol | bior_drill -p mapped_OMIM_ID | bior_lookup -d
$bior/omim/2013_02_27/genemap_GRCh37.tsv.bgz -p MIM_Number | bior_drill -p
Disorders
#UNKNOWN_1mapped_OMIM_IDDisorders
MRPL39611845
PANX2608421.
BRCA1113705{Breast-ovarian cancer, familial, 1}, 604370 (3); {Pancreatic
cancer,
```

Note: period '.' always means the value was not in the dataset. So in this case, some genes are not associated with disorders in OMIM.

## Generating an OMIM Disorder Report for a Set of rsIDs

Want OMIM

```
cat example.vcf | bior_vcf_to_tjson | bior_overlap ---d
$catalogs/NCBIGene/GRCh37_p10/ genes.tsv.bgz | bior_drill ---p GeneID ---p
gene ---p MIM | cut ---f9 --- ---complement | bior_lookup ---d
$catalogs/omim/2013_02_27/ genemap_GRCh37.tsv.bgz ---p MIM_Number |
bior_drill ---p Disorders > example.w_omim.
```

Use lookup to also find any disease/condition information in OMIM. First, the gene catalog just happens to have the OMIM id ("MIM"), so alter the command to drill that out:

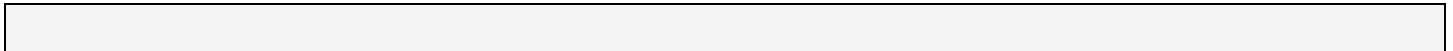
Want OMIM

```
cat example.vcf | bior_vcf_to_tjson | bior_overlap ---d
$catalogs/NCBIGene/GRCh37_p10/ genes.tsv.bgz | bior_drill ---p GeneID ---p
gene ---p MIM | cut ---f9 --- ---complement | bior_lookup ---d
$catalogs/omim/2013_02_27/ genemap_GRCh37.tsv.bgz ---p MIM_Number |
bior_drill ---p Disorders > example.w_omim.
```



```
$ cat example.vcf | bior_vcf_to_tjson | bior_overlap -d
$bior/NCBIGene/GRCh37_p10/genes.tsv.bgz | bior_drill -p GeneID -p gene | cut
-f 9 --complement | bior_lookup -d $bior/hgnc/2012_08_12/hgnc_GRCh37.tsv.bgz
-p Approved_Symbol

$ cat example.vcf | bior_vcf_to_tjson | bior_overlap -d
$bior/NCBIGene/GRCh37_p10/genes.tsv.bgz | bior_drill -p GeneID -p gene -p MIM
| cut -f 9 --complement
##fileformat=VCFv4.0
#CHROMPOSIDREFALTQUALFILTERINFOGeneIDgeneMIM
1215848808rs116645811GA...7399USH2A608400
1215848808rs116645811GT...7399USH2A608400
...
$
```



```
$ cat example.vcf | bior_vcf_to_tjson | bior_overlap -d
$bior/NCBIGene/GRCh37_p10/genes.tsv.bgz | bior_drill -p GeneID -p gene -p MIM
| cut -f 9 --complement | bior_lookup -d
$bior/omim/2013_02_27/genemap_GRCh37.tsv.bgz -p MIM_Number |
bior_pretty_print
# COLUMN NAME COLUMN VALUE
- - - - -
1 CHROM 1
2 POS 215848808
3 ID rs116645811
4 REF G
5 ALT A
6 QUAL .
7 FILTER .
8 INFO .
9 GeneID 7399
10 gene USH2A
11 MIM 608400
12 LookupPipe {
  "Chromosome.Map_Entry_Number": 1.1272,
  "MonthEntered": 1,
  "Day": 27,
  "Year": 4,
  "Cytogenetic_location": "1q41",
  "GeneSymbols": "USH2A, RP39",
  "Gene_Status": "C",
  "Title": "Usherin",
  "Title_cont": "",
  "MIM_Number": 608400,
  "Method": "Fd",
  "Comments": "",
  "Disorders": "Usher syndrome, type 2A, 276901 (3);
Retinitis pigmentosa 39, 613809",
  "Disorders_cont": " ",
  "Mouse_correlate": "1(Ush2a)"
}
$
```

Looks like we want the column "Disorders":

```
$ cat example.vcf | bior_vcf_to_tjson | bior_overlap -d
$bior/NCBIGene/GRCh37_p10/genes.tsv.bgz | bior_drill -p GeneID -p gene -p MIM
| cut -f 9 --complement | bior_lookup -d
$bior/omim/2013_02_27/genemap_GRCh37.tsv.bgz -p MIM_Number | bior_drill -p
Disorders
##fileformat=VCFv4.0
#CHROMPOSIDREFALTQUALFILTERINFOGeneIDgeneMIMDisorders
1215848808rs116645811GA...7399USH2A608400Usher syndrome, type 2A, 276901 (3);
Retinitis pigmentosa 39, 613809
...
2250616806rs5771206AG...56666PANX2608421.
$
```

OK, lets go and get some information from some variant catalogs that are not Allele frequencies:

First, dbSNP has all kinds of useful information including

"INFO.dbSNPBuildID":

"INFO.SSR": SSR 1 Integer 247,783 0.49% SNP Suspect Reason Code SNP Suspect Reason Code, 0 - unspecified, 1 - Paralog, 2 - byEST, 3 - Para\_EST, 4 - oldAlign, 5 - other. Count in column D is non-zero

Sequence Annotation Flags

"INFO.SCS": Integer 12,533 0.02% SNP Clinical Significance SNP Suspect Reason Code, 0 - unspecified, 1 - Paralog, 2 - byEST, 3 - Para\_EST, 4 - oldAlign, 5 - other. Count in column D is non-zero

"INFO.CLN": CLN 0 Flag 31,524 0.06% SNP is Clinical Includes

LSDB,OMIM,TPA,Diagnostic

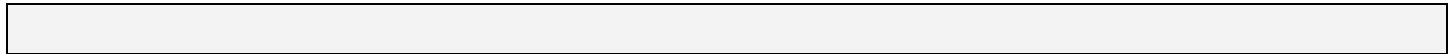
"INFO.SAO": SAO 1 Integer 14,908 0.03% SNP Allele Origin SNP Allele Origin: 0 - unspecified, 1 - Germline, 2 - Somatic, 3 - Both. Count in column D is non-zero

"\_id": The rs\_id, a (near)universal identifier for the Variant.

(to see a compiled list of what is in this, go to the bsi documentation: <http://bsiweb.mayo.edu/dbsnp>)

This text file is a good guide (downloaded from dbSNP:

[ftp://ftp.ncbi.nih.gov/snp/organisms/human\\_9606/VCF/00-snp\\_info\\_tags.txt](ftp://ftp.ncbi.nih.gov/snp/organisms/human_9606/VCF/00-snp_info_tags.txt))



```

7  FILTER      .
8  INFO        .
9  VCF2VariantPipe {
    "CHROM": "21",
    ...
  }
10 SameVariantPipe {
    "CHROM": "21",
    "POS": "26965148",
    "ID": "rs1135638",
    "REF": "G",
    "ALT": "A",
    "QUAL": ".",
    "FILTER": ".",
    "INFO": {
      "RSPOS": 26965148,
      "RV": true,
      "GMAF": 0.2395,
      "dbSNPBuildID": 86,
      "SSR": 0,
      "SAO": 0,
      "VP": "05030000030507051f000100",
      "GENEINFO": "MRPL39:54148",
      "WGT": 1,
      "VC": "SNV",
      "S3D": true,
      "SLO": true,
      "REF": true,
      "SYN": true,
      "ASP": true,
      "VLD": true,
      "G5A": true,
      "G5": true,
      "HD": true,
      "GNO": true,
      "KGPhasel": true,
      "KGPilot123": true,
      "KGPROD": true,
      "OTHERKG": true,
      "PH3": true
    },
    "_id": "rs1135638",

```

To match variants, use same\_variant:

Now build a table with: rs\_id, dbSNPBuildID, SSR, SCS, CLN, SAO, and CLN, do this:

```
$ cat example.vcf | bior_vcf_to_tjson | bior_same_variant -d
$bior/dbSNP/137/00-All_GRCh37.tsv.bgz | bior_drill -p _id -p dbSNPBuildID -p
INFO.SSR -p INFO.SCS -p INFO.CLN -p INFO.SAO -p INFO.CLN | cut -f 9 --
complement
```

unfortunately, the variants in this example file, did not have any results, as these annotations are rather sparse. Finding variants with these properties can be a trick. Here is a trick that I use to cat all variants from a specific gene:

```
$ zcat $bior/NCBIGene/GRCh37_p10/genes.tsv.bgz | grep "\"gene\": \"BRCA1\""
174119631241277500{"_type": "gene", "_landmark": "17", "_strand": "-", "_minBP":
41196312, "_maxBP": 41277500, "gene": "BRCA1", "gene_synonym": "BRCAI; BRCC1;
BROVCA1; IRIS; PNCA4; PPP1R53; PSCP; RNF53", "note": "breast cancer 1, early
onset; Derived by automated computational analysis using gene prediction
method: BestRefseq.", "GeneID": "672", "HGNC": "1100", "HPRD": "00218", "MIM": "
113705"}
$
```

Then to find a variant in dbSNP with an SAO annotation:

```

    _strand : "-",
    "_minBP": 41196312,
    "_maxBP": 41277500,
    "gene": "BRCA1",
    "gene_synonym": "BRCAI; BRCC1; BROVCA1; IRIS; PNCA4;
PPP1R53; PSCP; RNF53",
    "note": "breast cancer 1, early onset; Derived by automated
computational analysis using gene prediction method: BestRefseq.",
    "GeneID": "672",
    "HGNC": "1100",
    "HPRD": "00218",
    "MIM": "113705"
  }
5 #UNKNOWN_5 {
  "CHROM": "17",
  "POS": "41196363",
  "ID": "rs8176320",
  "REF": "C",
  "ALT": "T",
  "QUAL": ".",
  "FILTER": ".",
  "INFO": {
    "RSPOS": 41196363,
    "RV": true,
    "GMAF": 0.0050,
    "dbSNPBuildID": 117,
    "SSR": 0,
    "SAO": 0,
    "VP": "050000800201040517000100",
    "GENEINFO": "BRCA1:672",
    "WGT": 1,
    "VC": "SNV",
    "REF": true,
    "U3": true,
    "VLD": true,
    "HD": true,
    "GNO": true,
    "KGPhasel": true,
    "KGPROD": true,
    "OTHERKG": true,
    "PH3": true
  },

```

COSMIC:

```

1  CHROM      21
2  POS        40190405
3  ID         rs115908228
4  REF        G
5  ALT        A
6  QUAL       .
7  FILTER     .
8  INFO       .
9  VCF2VariantPipe {
    "CHROM": "21",
...
}
10 SameVariantPipe {
    "Gene_name": "ETS2",
    "Accession_Number": "ENST00000360214",
    "HGNC_ID": "3489",
    "Sample_name": "107702",
    "ID_sample": "1520464",
    "ID_tumour": "1442839",
    "Primary_site": "breast",
    "Site_subtype": "NS",
    "Primary_histology": "carcinoma",
    "Histology_subtype": "HER-positive_carcinoma",
    "Genome-wide_screen": "n",
    "Mutation_ID": "94254",
    "Mutation_CDS": "c.646G\u003eA",
    "Mutation_AA": "p.G216S",
    "Mutation_Description": "Substitution - Missense",
    "Mutation_GRCh37_genome_position": "21:40190405-
40190405",
    "Mutation_GRCh37_strand": "+",
    "Mutation_somatic_status": "Confirmed somatic
variant",
    "Pubmed_PMID": "20668451",
    "Sample_source": "NS",
    "Tumour_origin": "primary",
    "_type": "variant",
    "_landmark": "21",
    "_refAllele": "G",
    "_altAlleles": [
        "A"
    ],

```

```
$ cat example.vcf | bior_vcf_to_tjson | bior_same_variant -d
$ bior/cosmic/v63/CosmicCompleteExport_GRCh37.tsv.bgz | bior_drill -p
Mutation_ID -p Mutation_CDS -p Mutation_AA -p Mutation_GRCh37_strand | cut -f
9 --complement
##fileformat=VCFv4.0
#CHROMPOSIDREFALTQUALFILTERINFO Mutation_ID Mutation_CDS Mutation_AA Mutation_GRC
h37_strand
121584880 rs116645811 GA.....
121584880 rs116645811 GT.....
...
214019040 rs115908228 GA...94254c.646G>Ap.G216S+
...
223085737 rs2240345 AC...330401c.1005T>Gp.D335E-
...
223962179 rs35978693 GT...39683c.657C>Ap.P219P-
$
```

Want UCSC Tracks (blacklisted) `cat example.vcf | bior_vcf_to_tjson | bior_overlap --d $catalogs/ucsc/hg19/wgEncodeDacMapabilityConsensusExcludable_GR`

`Ch37.tsv.bgz | bior_drill --p score | complement > example.w_ucsc.vcf`

## UCSC:

The UCSC catalogs related to TREAT are the following:

```
export ucsc=$bior/ucsc/ ;
export blacklistedFile=$ucsc/hg19/wgEncodeDacMapabilityConsensusExcludable_GRCh37.tsv.bgz ;
export repeatFile=$ucsc/hg19/rmsk_GRCh37.tsv.bgz ;
export regulationFile=$ucsc/hg19/oreganno_GRCh37.tsv.bgz ;
export uniqueFile=$ucsc/hg19/wgEncodeDukeMapabilityRegionsExcludable_GRCh37.tsv.bgz ;
export tssFile=$ucsc/hg19/switchDbTss_GRCh37.tsv.bgz ;
export tfbsFile=$ucsc/hg19/tfbsConsSites_GRCh37.tsv.bgz ;
export enhancerFile=$ucsc/hg19/vistaEnhancers_GRCh37.tsv.bgz ;
export conservationFile=$ucsc/hg19/phastConsElements46wayPrimates_GRCh37.tsv.bgz ;
```

To annotate with any of these files, do something like this:



```
$ cat example.vcf | bior_vcf_to_tjson | bior_overlap -d $blacklistedFile |
  bior_drill -p score | cut -f 9 --complement
##fileformat=VCFv4.0
#CHROMPOSIDREFALTQUALFILTERINFOscore
1215848808rs116645811GA....
1215848808rs116645811GT....
1215848808rs116645811GG....
1215848808rs116645811GC....
...
```

unfortunately, our example file does not overlap many of these rare features. Another way to think about this is "what genes of interest overlap some UCSC genomic feature".

```
$ zcat $bior/NCBIGene/GRCh37_p10/genes.tsv.bgz | bior_overlap -d
$blacklistedFile | grep -v "{}" | bior_drill -c -2 -p gene | cut -f 5
gene
MTND1P23
MTND2P28
TTC34
RNU1-1
RSP01
HFM1
AMY2A
NOTCH2NL
NBPF17P
PMF1
PMF1-BGLAP
PCNXL2
RYSR2
MTND2P27
```

This list of genes could then be used in a lookup query later, or you could cut the JSON instead of the gene name and use that to overlap the data in your VCF file in a filtering process.

A similar technique can be use to pair down the variants based on those variants that you do NOT want because overlapping some genomic feature would indicate it is unlikely to be significant.

## Putting it all Together – Making a Genomic Feature Annotation Program

Below is a simple example of an annotation program using the simple scripts.

```

$ cat treatGF.bior
bior_vcf_to_tjson < /dev/stdin \
| bior_overlap -d $bior/NCBIGene/GRCh37_p10/genes.tsv.bgz \
| bior_drill -p gene -p GeneID -p MIM \
| bior_lookup -d $bior/hgnc/2012_08_12/hgnc_GRCh37.tsv.bgz -p Approved_Symbol
-c -3 \
| bior_drill -p Approved_Symbol -p Entrez_Gene_ID -p Ensembl_Gene_ID -p
UniProt_ID \
| bior_lookup -d $bior/omim/2013_02_27/genemap_GRCh37.tsv.bgz -p MIM_Number -
c -5 \
| bior_drill -p Disorders \
| bior_overlap -d $bior/mirbase/release19/hsa_GRCh37.p5.tsv.bgz -c -9 \
| bior_drill -p ID \
| bior_overlap -d
$bior/ucsc/hg19/wgEncodeDacMapabilityConsensusExcludable_GRCh37.tsv.bgz -c
-10 \
| bior_drill -p score \
| bior_overlap -d $bior/ucsc/hg19/phastConsElements46way_GRCh37.tsv.bgz -c
-11 \
| bior_drill -p score \
| bior_overlap -d $bior/ucsc/hg19/oreganno_GRCh37.tsv.bgz -c -12 \
| bior_drill -p score \
| bior_overlap -d $bior/ucsc/hg19/tfbsConsSites_GRCh37.tsv.bgz -c -13 \
| bior_drill -p score \
| bior_overlap -d $bior/ucsc/hg19/switchDbTss_GRCh37.tsv.bgz -c -14 \
| bior_drill -p score \
| bior_overlap -d $bior/ucsc/hg19/vistaEnhancers_GRCh37.tsv.bgz -c -15 \
| bior_drill -p score \
| bior_overlap -d
$bior/ucsc/hg19/wgEncodeDukeMapabilityRegionsExcludable_GRCh37.tsv.bgz -c -16
\
| bior_drill -p score \
| bior_overlap -d $bior/ucsc/hg19/rmsk_GRCh37.tsv.bgz -c -17 \
| bior_drill -p score \
| bior_overlap -d
$bior/ucsc/hg19/wgEncodeDukeMapabilityRegionsExcludable_GRCh37.tsv.bgz -c -18
\
| bior_drill -p score \
| ./removeJSON.pl
$

```

## 6. Examples Matching Alleles (bior\_same\_variant)

Allele Frequencies:

on the RCF:

BGI:

```

5 ALT      A
6 QUAL      .
7 FILTER      .
8 INFO      .
9 VCF2VariantPipe {
    "CHROM": "21",
    "POS": "26965148",
    "ID": "rs1135638",
    "REF": "G",
    "ALT": "A",
    "QUAL": ".",
    "FILTER": ".",
    "INFO": {
        ".": true
    },
    "_id": "rs1135638",
    "_type": "variant",
    "_landmark": "21",
    "_refAllele": "G",
    "_altAlleles": [
        "A"
    ],
    "_minBP": 26965148,
    "_maxBP": 26965148
}
10 SameVariantPipe {
    "chromosome_id": "chr21",
    "genomic_position": 25887019,
    "index_of_major_allele": 0,
    "major_allele": "A",
    "index_of_minor_allele": 2,
    "minor_allele": "G",
    "number_A": 710,
    "number_C": 1,
    "number_G": 428,
    "number_T": 2,
    "estimated_minor_allele_freq": 0.278705,
    "estimated_major_allele_freq": 0.721295,
    "is_in_dbSNP": 1,
    "_landmark": "21",
    "_refAllele": "G",
    "_altAlleles": [

```

```
$ cat example.vcf | bior_vcf_to_tjson | bior_same_variant -d
$bior/BGI/hg19/LuCAMP_200exomeFinal.maf_GRCh37.tsv.bgz | bior_drill -p
estimated_major_allele_freq -p estimated_minor_allele_freq | cut --complement
-f 9
...
2230823196rs5753130TC...0.5765180.423482
2230856121rs35764129GA...0.9573590.042641
2230857373rs2240345AC...0.6109330.389067
2230857448rs5749104AG...0.5872320.412768
2230857645rs114917409CG.....
2230858149rs115111929AC.....
2230860830rs2269961CT...0.8081760.191824
...
```

dbSNP:

--

```

"FILTER": ".",
"INFO": {
  ".": true
},
"_id": "rs1135638",
"_type": "variant",
"_landmark": "21",
"_refAllele": "G",
"_altAlleles": [
  "A"
],
"_minBP": 26965148,
"_maxBP": 26965148
}

```

```

10 SameVariantPipe {
  "CHROM": "21",
  "POS": "26965148",
  "ID": "rs1135638",
  "REF": "G",
  "ALT": "A",
  "QUAL": ".",
  "FILTER": ".",
  "INFO": {
    "RSPOS": 26965148,
    "RV": true,
    "GMAF": 0.2395,
    "dbSNPBuildID": 86,
    "SSR": 0,
    "SAO": 0,
    "VP": "05030000030507051f000100",
    "GENEINFO": "MRPL39:54148",
    "WGT": 1,
    "VC": "SNV",
    "S3D": true,
    "SLO": true,
    "REF": true,
    "SYN": true,
    "ASP": true,
    "VLD": true,
    "G5A": true,
    "G5": true,
    "HD": true,

```

```

"FILTER": ".",
"INFO": {
  ".": true
},
"_id": "rs1135638",
"_type": "variant",
"_landmark": "21",
"_refAllele": "G",
"_altAlleles": [
  "A"
],
"_minBP": 26965148,
"_maxBP": 26965148
}

```

```

10 SameVariantPipe {
  "CHROM": "21",
  "POS": "26965148",
  "ID": "rs1135638",
  "REF": "G",
  "ALT": "A",
  "QUAL": ".",
  "FILTER": ".",
  "INFO": {
    "RSPOS": 26965148,
    "RV": true,
    "GMAF": 0.2395,
    "dbSNPBuildID": 86,
    "SSR": 0,
    "SAO": 0,
    "VP": "05030000030507051f000100",
    "GENEINFO": "MRPL39:54148",
    "WGT": 1,
    "VC": "SNV",
    "S3D": true,
    "SLO": true,
    "REF": true,
    "SYN": true,
    "ASP": true,
    "VLD": true,
    "G5A": true,
    "G5": true,
    "HD": true,

```

dbSNP:

```
##fileformat=VCFv4.0
#CHROMPOSIDREFALTQUALFILTERINFOINFO.dbSNPBuildIDINFO.SSRINFO.SCSINFO.CLNINFO.
SAO_id
1215848808rs116645811GA.....
1215848808rs116645811GT.....
1215848808rs116645811GG.....
1215848808rs116645811GC.....
1215848808rs116645811CA.....
...
$
```

ESP:

--



```

],
"AA_AC": [
  "3307",
  "1099"
],
"TAC": [
  "10418",
  "2588"
],
"MAF": [
  "17.314",
  "24.9433",
  "19.8985"
],
"GTS": [
  "AA",
  "AG",
  "GG"
],
"EA_GTC": [
  "2954",
  "1203",
  "143"
],
"AA_GTC": [
  "1229",
  "849",
  "125"
],
"GTC": [
  "4183",
  "2052",
  "268"
],
"DP": 75,
"GL": [
  "MRPL39"
],
"CP": 1.0,
"CG": 3.0,
"AA": "A",
"CA": [

```

HapMap:

```
6  QUAL      .
7  FILTER    .
8  INFO      .
9  VCF2VariantPipe {
    "CHROM": "21",
    "POS": "26965148",
    "ID": "rs1135638",
    "REF": "G",
    "ALT": "A",
    "QUAL": ".",
    "FILTER": ".",
    "INFO": {
        ".": true
    },
    "_id": "rs1135638",
    "_type": "variant",
    "_landmark": "21",
    "_refAllele": "G",
    "_altAlleles": [
        "A"
    ],
    "_minBP": 26965148,
    "_maxBP": 26965148
}
10 SameVariantPipe {
    "rsNumber": "rs1135638",
    "chrom": "chr21",
    "pos": 25887019,
    "strand": "+",
    "build": "ncbi_b36",
    "refallele": "G",
    "otherallele": "A",
    "_type": "variant",
    "_landmark": "21",
    "_minBP": 26965148,
    "_maxBP": 26965148,
    "_strand": "+",
    "_refAllele": "G",
    "_altAlleles": [
        "A"
    ],
    "id": "rs1135638",
```



```

9   VCF2VariantPipe {
    "CHROM": "21",
    "POS": "26965148",
    "ID": "rs1135638",
    "REF": "G",
    "ALT": "A",
    "QUAL": ".",
    "FILTER": ".",
    "INFO": {
      ".": true
    },
    "_id": "rs1135638",
    "_type": "variant",
    "_landmark": "21",
    "_refAllele": "G",
    "_altAlleles": [
      "A"
    ],
    "_minBP": 26965148,
    "_maxBP": 26965148
  }
10  SameVariantPipe {
    "rsNumber": "rs1135638",
    "chrom": "chr21",
    "pos": 25887019,
    "strand": "+",
    "build": "ncbi_b36",
    "refallele": "G",
    "otherallele": "A",
    "_type": "variant",
    "_landmark": "21",
    "_minBP": 26965148,
    "_maxBP": 26965148,
    "_strand": "+",
    "_refAllele": "G",
    "_altAlleles": [
      "A"
    ],
    "_id": "rs1135638",
    "CEU": {
      "center": "sanger",
      "protLSID": "urn:LSID:illumina.hapmap.org:Protocol:

```

```

},
"CHD": {
  "center": "sanger",
  "protLSID": "urn:LSID:illumina.hapmap.org:Protocol:Human_1M_BeadChip:3",
  "assayLSID": "urn:LSID:sanger.hapmap.org:Assay:H1Mrs1135638:3",
  "panellSID": "urn:lsid:dcc.hapmap.org:Panel:US_Chinese:4",
  "QC_code": "QC+",
  "refallele_freq": 0.289,
  "refallele_count": 63,
  "otherallele_freq": 0.711,
  "otherallele_count": 155,
  "totalcount": 218
},
"GIH": {
  "center": "sanger",
  "protLSID": "urn:LSID:illumina.hapmap.org:Protocol:Human_1M_BeadChip:3",
  "assayLSID": "urn:LSID:sanger.hapmap.org:Assay:H1Mrs1135638:3",
  "panellSID": "urn:lsid:dcc.hapmap.org:Panel:US_Gujarati:4",
  "QC_code": "QC+",
  "refallele_freq": 0.49,
  "refallele_count": 97,
  "otherallele_freq": 0.51,
  "otherallele_count": 101,
  "totalcount": 198
},
"MEX": {
  "center": "sanger",
  "protLSID": "urn:LSID:illumina.hapmap.org:Protocol:Human_1M_BeadChip:3",
  "assayLSID": "urn:LSID:sanger.hapmap.org:Assay:H1Mrs1135638:3",
  "panellSID": "urn:lsid:dcc.hapmap.org:Panel:US_Mexican-30-trios:4",
  "QC_code": "QC+",
  "refallele_freq": 0.237,
  "refallele_count": 27,
  "otherallele_freq": 0.763,
  "otherallele_count": 87,
  "totalcount": 114
},
"YRI": {
  "center": "sanger",
  "protLSID": "urn:LSID:illumina.hapmap.org:Protocol:Human_1M_BeadChip:3",
  "assayLSID": "urn:LSID:sanger.hapmap.org:Assay:H1Mrs1135638:3",
  "panellSID": "urn:lsid:dcc.hapmap.org:Panel:Yoruba-60-trios:4",

```

```

},

```

```

        "protLSID": "urn:LSID:illumina.hapmap.org:Protocol:
Human_1M_BeadChip:3",
        "assayLSID": "urn:LSID:sanger.hapmap.org:Assay:
H1Mrs1135638:3",
        "panelLSID": "urn:lsid:dcc.hapmap.org:Panel:Italian:
4",
        "QC_code": "QC+",
        "refallele_freq": 0.201,
        "refallele_count": 41,
        "otherallele_freq": 0.799,
        "otherallele_count": 163,
        "totalcount": 204
    },
    "JPT": {
        "center": "sanger",
        "protLSID": "urn:LSID:illumina.hapmap.org:Protocol:
Human_1M_BeadChip:3",
        "assayLSID": "urn:LSID:sanger.hapmap.org:Assay:
H1Mrs1135638:3",
        "panelLSID": "urn:lsid:dcc.hapmap.org:Panel:
Japanese:4",
        "QC_code": "QC+",
        "refallele_freq": 0.339,
        "refallele_count": 76,
        "otherallele_freq": 0.661,
        "otherallele_count": 148,
        "totalcount": 224
    },
    "LWK": {
        "center": "sanger",
        "protLSID": "urn:LSID:illumina.hapmap.org:Protocol:
Human_1M_BeadChip:3",
        "assayLSID": "urn:LSID:sanger.hapmap.org:Assay:
H1Mrs1135638:3",
        "panelLSID": "urn:lsid:dcc.hapmap.org:Panel:
Luhya_Kenyan:4",
        "QC_code": "QC+",
        "refallele_freq": 0.323,
        "refallele_count": 71,
        "otherallele_freq": 0.677,
        "otherallele_count": 149,
        "totalcount": 220
    }
}

```

```

$ cat example.vcf | bior_vcf_to_tjson | bior_same_variant -d
$ bior/hapmap/2010-08_phaseII+III/allele_freqs_GRCh37.tsv.bgz | bior_drill -p
CEU.refallele_freq -p CEU.otherallele_freq -p YRI.refallele_freq -p YRI.
otherallele_freq -p JPT.refallele_count -p JPT.otherallele_count -p JPT.
totalcount -p CHB.refallele_count -p CHB.otherallele_count -p CHB.totalcount
| cut --complement -f 9
##fileformat=VCFv4.0
#CHROMPOSIDREFALTQUALFILTERINFOCEU.refallele_freqCEU.otherallele_freqYRI.
refallele_freqYRI.otherallele_freqJPT.refallele_countJPT.
otherallele_countJPT.totalcountCHB.refallele_countCHB.otherallele_countCHB.
totalcount
1215848808rs116645811GA.....
1215848808rs116645811GT.....
1215848808rs116645811GG.....
1215848808rs116645811GC.....
1215848808rs116645811CA.....
1215848808rs116645811CT.....
1215848808rs116645811CG.....
1215848808rs116645811CC.....
1215848808rs116645811AA.....
1215848808rs116645811AT.....
1215848808rs116645811AG.....
1215848808rs116645811AC.....
1215848808rs116645811TA.....
1215848808rs116645811TT.....
1215848808rs116645811TG.....
1215848808rs116645811TC.....
2126965148rs1135638GA...0.1770.8230.2690.7317614822474192266
2126965172rs010576TC.....
2126965205rs1057885TC...0.1540.8460.2380.762305686265884
2126976144rs116331755AG.....
2126976222rs7278168CT...1.000.7390.261761086791190
2126976237rs7278284CT.....
2126978790rs75377686TC.....
2126978950rs3989369AG...0.0350.9650.2650.735222422610264274
...

```

1000 Genomes:



```

9   varzvariantPipe {
    "CHROM": "21",
    "POS": "26965148",
    "ID": "rs1135638",
    "REF": "G",
    "ALT": "A",
    "QUAL": ".",
    "FILTER": ".",
    "INFO": {
      ".": true
    },
    "_id": "rs1135638",
    "_type": "variant",
    "_landmark": "21",
    "_refAllele": "G",
    "_altAlleles": [
      "A"
    ],
    "_minBP": 26965148,
    "_maxBP": 26965148
  }
10 SameVariantPipe {
    "CHROM": "21",
    "POS": "26965148",
    "ID": "rs1135638",
    "REF": "G",
    "ALT": "A",
    "QUAL": "100",
    "FILTER": "PASS",
    "INFO": {
      "AVGPOST": 1.0,
      "RSQ": 0.9999,
      "SNPSOURCE": [
        "LOWCOV",
        "EXOME"
      ],
      "AN": 2184,
      "LDAF": 0.7609,
      "VT": "SNP",
      "AA": "A",
      "AC": [
        1661

```

```
"_altAlleles":[
    "A"
],
"_minBP": 26965148,
"_maxBP": 26965148
}
$
```

```
$ cat example.vcf | bior_vcf_to_tjson | bior_same_variant -d
$bior/1000_genomes/20110521/ALL.wgs.phase1_release_v3.20101123.
snps_indels_sv.sites_GRCh37.tsv.gz | bior_drill -p INFO.AF -p INFO.EUR_AF -p
INFO.ASN_AF -p INFO.AFR_AF -p INFO.AMR_AF | cut -f 9 --complement
##fileformat=VCFv4.0
#CHROMPOSIDREFALTQUALFILTERINFOINFO.AFINFO.EUR_AFINFO.ASN_AFINFO.AFR_AFINFO.
AMR_AF
1215848808rs116645811GA.....
...
1215848808rs116645811TC.....
2126965148rs1135638GA...0.760.80.710.720.8
2126965172rs010576TC...0.01..0.040.01
2126965205rs1057885TC...0.760.80.710.720.8
2126976144rs116331755AG...9.0E-4..0.0041.
2126976222rs7278168CT...0.110.00260.140.240.14
2126976237rs7278284CT...0.120.00260.140.270.14
2126978790rs75377686TC...0.01..0.040.01
2126978950rs3989369AG...0.910.960.970.750.94
...
$
```

## Putting it All Together Building an AF Pipeline

```
TREAT]$ cat treatAF.bior
export bior=$bior/
cat /dev/stdin | bior_vcf_to_tjson \
| bior_same_variant -d $bior/dbSNP/137/00-All_GRCh37.tsv.bgz \
| bior_drill -p _id -p INFO.dbSNPBuildID -p INFO.SSR -p INFO.SCS -p INFO.CLN
-p INFO.SAO \
| bior_same_variant -c -7 -d $bior/cosmic/v63/CosmicCompleteExport_GRCh37.
tsv.bgz \
| bior_drill -p Mutation_ID -p Mutation_CDS -p Mutation_AA -p
Mutation_GRCh37_strand \
| bior_same_variant -c -11 -d $bior/1000_genomes/20110521/ALL.wgs.
phase1_release_v3.20101123.snps_indels_sv.sites_GRCh37.tsv.gz \
| bior_drill -p INFO.ASN_AF -p INFO.AMR_AF -p INFO.AFR_AF -p INFO.EUR_AF \
| bior_same_variant -c -15 -d $bior/BGI/hg19/LuCAMP_200exomeFinal.maf_GRCh37.
tsv.bgz \
| bior_drill -p estimated_minor_allele_freq \
| bior_same_variant -c -16 -d $bior/ESP/build37/ESP6500SI_GRCh37.tsv.bgz \
| bior_drill -p INFO.MAF[0] -p INFO.MAF[1] -p INFO.MAF[2] \
| bior_same_variant -c -19 -d $bior/hapmap/2010-
08_phaseII+III/allele_freqs_GRCh37.tsv.bgz \
| bior_drill -p CEU.refallele_freq -p CEU.otherallele_freq \
| ./removeJSON.pl
TREAT]$
```

## 7. Extracting Data with JSONPaths (bior\_drill)

To extract data that is embedded in a JSON document as an array you can use `drill.path[1]` to get the first element in the array, `drill.path[1].field` to get a field in a json array or `drill.path[*]` to get all elements in the array.

## 8. Command Line Tools

Want SNPeff

```
cat example.vcf | bior_snpeff | bior_drill -p Effect -p Effect_impact -p
Functional_class -p Amino_acid_change | cut ---f 9 --- ---complement >
example.w_genes.vcf
```

Want SIFT & PolyPhen

```
cat example.vcf | bior_vep | bior_drill -p Consequence -p SIFT -p PolyPhen -p
SIFT Score -p PolyPhen Score | cut -f 9 --- ---complement > example.w genes.
vcf

TREAT]$ cat treatTOOLS.bior
bior_vep < /dev/stdin \
| bior_drill -p Allele -p Gene -p Feature -p Feature_type -p Consequence -p
cDNA_position -p CDS_position -p Protein_position -p Amino_acids -p Codons -p
HGNC -p SIFT_TERM -p SIFT_Score -p PolyPhen_TERM -p PolyPhen_Score \
| bior_snpeff \
| bior_drill -p Effect -p Effect_impact -p Functional_class -p Codon_change -
p Amino_acid_change -p Gene_name -p Gene_biotype -p Coding -p Transcript -p
Exon
TREAT]$
```

## 9. Mixing In Scripts and Languages

To find all overlapping genes that are not the same gene:

```
zcat $bior/NCBIGene/GRCh37_p10/genes.tsv.bgz | bior_overlap -d
$bior/v1/NCBIGene/GRCh37_p10/genes.tsv.bgz | perl -e 'while (<>) {chomp;
@a=split(/\t/, $_); if($a[3] ne $a[4]){print $a[3]."\t".$a[4]."\n";} }' |
bior_drill -c -2 -p gene | bior_drill -c -2 -p gene | less
```

## 10. Common Problems

### Handling VCF Files with VERY large headers

All BioR commands store the header in memory. This is done because commands like `bior_vcf_to_tjson` use the header to understand the structure of the data lines and parse the lines into JSON more intelligently (e.g. identify numbers instead of strings, identify arrays, etc.). In production, we have noticed that some headers are extremely large (multiple megabytes). When a user runs BioR, the header is expanded into objects in memory for each BioR command. This can lead to BioR slowing to a crawl when the ram on the machine is exceeded. Internally what happens is that the header is chopped off and stored in memory, then each row streams through the system as an array of strings. The data rows are not that large, but the metadata in the header may get copied many times in memory as transformations are done on the data. The best workaround for this problem is to use `grep` to cut off all excess header lines (e.g. lines that are not descriptive) then push the BioR output on to the file. Recombine the header if needed.

e.g.

```
zcat example.vcf.gz | head -n 10000 | grep -v "###" > mylongheader.vcf
```

```
zcat example.vcf.gz | bior_vcf_to_tjson | bior_mycommands >> mylongheader.vcf
```

### Large Memory Requirements

Sometimes users complain about large memory requirements from BioR – especially SNPEff. SNPEff,

when run in production requires 4Gb of Ram. BioR will align large insertions and deletions prior to sending them to SNPEff using the same exact method used in SNPEff. When processing these large variants, both BioR and SNPEff can crash. The current work-around for dealing with large variants is to pre-screen them and filter them out to another file prior to annotating with SNPEff. Hopefully the BioR team will be able to collect better statistics and not align large variants in the future.

### BioR exits with some error I don't understand

Rerun the same exact command with logging enabled (-l) and submit both the input file, and the results of the log to the BioR team. We will try to help you ASAP.

## 11. Creating Catalogs

### Indexing your Samples

Lets say you want to get variants in your sample that overlap a gene. One way to do this is to stream the variants e.g:

```
> cat example.vcf | head
##fileformat=VCFv4.0
#CHROM POS ID REF ALT QUAL FILTER INFO
21 26960070 rs116645811 G A . . .
21 26965148 rs1135638 G A . . .
21 26965172 rs010576 T C . . .
21 26965205 rs1057885 T C . . .
21 26976144 rs116331755 A G . . .
21 26976222 rs7278168 C T . . .
21 26976237 rs7278284 C T . . .
21 26978790 rs75377686 T C . . .
>cat example.vcf | bior_vcf_to_tjson | bior_overlap -d
$bior/NCBIGene/GRCh37_pl0/genes.tsv.bgz | grep "\"gene\": \"PANX2\""
22 50616005 rs35195493 C G . . . {"CHROM":"22","POS":"50616005","ID":"
rs35195493","REF":"C","ALT":"G","QUAL":".", "FILTER":".", "INFO":{".":true}, "
_id":"rs35195493","_type":"variant","_landmark":"22","_refAllele":"C","
_altAlleles":["G"],"_minBP":50616005,"_maxBP":50616005} {"_type":"gene","
_landmark":"22","_strand":"+","_minBP":50609160,"_maxBP":50618724,"gene":"
PANX2","gene_synonym":"hPANX2; PX2","note":"pannexin 2; Derived by automated
computational analysis using gene prediction method: BestRefseq.", "GeneID":"
56666","HGNC":"8600","HPRD":"09760","MIM":"608421"}
22 50616806 rs5771206 A G . . . {"CHROM":"22","POS":"50616806","ID":"
rs5771206","REF":"A","ALT":"G","QUAL":".", "FILTER":".", "INFO":{".":true}, "
_id":"rs5771206","_type":"variant","_landmark":"22","_refAllele":"A","
_altAlleles":["G"],"_minBP":50616806,"_maxBP":50616806} {"_type":"gene","
_landmark":"22","_strand":"+","_minBP":50609160,"_maxBP":50618724,"gene":"
PANX2","gene_synonym":"hPANX2; PX2","note":"pannexin 2; Derived by automated
computational analysis using gene prediction method: BestRefseq.", "GeneID":"
56666","HGNC":"8600","HPRD":"09760","MIM":"608421"}
$
```

If you just want variants that overlap any gene, you can always do something like:

```
>zcat $bior/NCBIGene/
GRCh37_p10/genes.tsv.bgz | bior_overlap -d ./example.tsv.gz |
grep -v "{}" | less
```

That works fine for a single gene, but what if you are starting with a list of genes? e.g.

```
>cat mygenes.txt
MRPL39
PANX2
BRCA1
...
```

In this case you may want to use an index on your data. To create the index, do something like:

```
>cat example.vcf | bior_vcf_to_tjson | grep "^#" | cut -f 1,2,9 |
  bior_drill -k -p _maxBP > example.tsv
>sort -k1,1 -k2,2n example.tsv
>bgzip example.tsv
>tabix example.tsv.gz
>tabix -s 1 -b 2 -e 3 example.tsv.gz
```

Now use lookup to get the gene locations, and overlap to overlap those locations with your data:

```
>cat mygenes.txt | bior_lookup -p gene
-d $bior/NCBIGene/GRCh37_p10/genes.tsv.bgz |
bior_overlap -d ./example.tsv.gz | bior_pretty_print
```

You can now use `bior_same_variant` to annotate variants that overlap your genes.

## Creating Custom Catalogs

One of the most powerful things about BioR is that users can publish their own catalogs and integrate new data into the system. They can also share these catalogs with others making the system extensible and much more powerful than a system where the catalogs must all be maintained by a single annotation team.

## The Publication Process

Publishing a catalog requires (1) a parser that understands arbitrarily formatted file formats, and (2) indexing tools. Parsers convert arbitrary data representations into JSON with a set of 'golden identifiers' the BioR system understands. Example 'golden identifiers' include `_landmark`, `_minBP`, and `_maxBP`. 'Golden identifiers' are always prefixed with an underscore ('\_') and must be absolutely consistent at both in terms of syntax and semantics. For example, `_minBP` uses the standard 1-based coordinate system (e.g. NCBI/Blast) not interbase coordinates

([http://gmod.org/wiki/Introduction\\_to\\_Chado#Interbase\\_Coordinates](http://gmod.org/wiki/Introduction_to_Chado#Interbase_Coordinates)), and `_strand` is represented as '+', '-', or '.' and NOT 'complement' as in the gbs files from NCBI. One of the functions of a parser, is to convert from arbitrary file formats into JSON, the other is to extract the 'golden identifiers' and place them in the JSON. 'Golden identifiers' are created so that BioR programs (e.g. `bior_overlap.sh`) can work on the information regardless of the source file format (e.g. VCF, GFF, GBS, XML, RelationalDB, Tab-Delimited, ...).

As they become available, parsers, will be exposed to users as command line tools. For example, `bior_vcf_to_variants.sh` is a parser that converts vcf to BioR JSON.

In summary, to make a custom catalog, you need:

1. Columns 1-3 bed-like (chr start stop) [1-based]
2. The 4<sup>th</sup> column is a series of key-value pairs enclosed by quotes and brackets
3. The 4 column contains "Golden identifiers" [`_landmark`, `_minBP`, and `_maxBP`]

Once this is created, use `bgzip` & `tabix` to compress and index it for genomic search. For those samples that do NOT have a genomic position, use the following values (`bior_create_catalog` will do this for you).

Golden Identifier	Default Value
<code>landmark</code>	UNKNOWN ( a period '.' is also ok)
<code>minBP</code>	0
<code>maxBP</code>	0

Zero is important because it has to be an integer and must be greater than zero. The JSON does not have to have the golden attribute if you won't search on it.

## Parsing and Converting the Data

If a parser for the file format is available (e.g. `bior_vcf_to_tjson`, `bior_bed_to_tjson`, ect.) publishing a custom catalog is extremely easy. Using the standard BioR tools, a publication pipeline can be constructed rapidly. For example:

```
zcat 00-All.vcf.gz | bior_vcf_to_tjson.sh | cut -f 9 | bior_drill.sh -k -p  
_landmark -p _minBP -p _maxBP > dbSNP.tsv
```

This pipeline streams the original VCF file past the parser (bior\_vcf\_to\_tjson), removes the content of the original VCF (cut -f 9) - this is ok, as all of this information is duplicated in the JSON format, drill out the key attributes (bior\_drill.sh) so that they can be indexed, and then output to a raw data file (dbSNP.tsv). The raw output file should look like this:

```
$ head dbSNP.tsv  
1 10144 10145 {"CHROM":"1","POS":"10144","ID":"rs144773400","REF":"  
TA","ALT":"T","QUAL":".", "FILTER":".", "INFO":{"RSPOS":10145,"dbSNPBuildID":  
134,"SSR":0,"SAO":0,"VP":"0500000000005000002000200","WGT":1,"VC":"DIV","ASP":  
true,"OTHERKG":true},"_id":"rs144773400","_type":"variant","_landmark":"1","  
_refAllele":"TA","_altAlleles":["T"],"_minBP":10144,"_maxBP":10145}  
1 10177 10177 {"CHROM":"1","POS":"10177","ID":"rs201752861","REF":"  
A","ALT":"C","QUAL":".", "FILTER":".", "INFO":{"RSPOS":10177,"dbSNPBuildID":  
137,"SSR":0,"SAO":0,"VP":"0500000000005000002000100","WGT":1,"VC":"SNV","ASP":  
true,"OTHERKG":true},"_id":"rs201752861","_type":"variant","_landmark":"1","  
_refAllele":"A","_altAlleles":["C"],"_minBP":10177,"_maxBP":10177}  
...
```

## Indexing the Data for Coordinate Based Search

For positional search, BioR supports indexing using Tabix. Tabix/bgzip should be installed in the RCF environment. First, compress the raw input. Assuming it is sorted:

```
$ bgzip dbSNP.tsv
```

Then run the tabix command:

```
$ tabix -s 1 -b 2 -e 3 dbSNP.tsv.gz &
```

That's it! you can now use your custom catalog as a database in BioR commands (e.g. bior\_overlap.sh -d /path/to/your/database.tsv.gz).

## Hints on Creating Indexes on Custom Catalogs

In addition to coordinate based search, users may also want to search a custom catalog based on IDs. The process is exactly the same as in indexing a catalog



described earlier in this document, but there are some gotcha's that users need to be aware of.

1. The catalog structure will not automatically join data. This can be frustrating as the data provider may not give the data to you in a desirable form (e.g. you may want to know everything the data provider knows about a gene, but they may have their data organized by variant or drug) so you will have to 'flip' the data around so that all information about a gene can be provided to users of your catalog. The BioR team has done this many times, and for Java programmers, there is a robust library (BioR-Catalog) and examples to help in the publication of new-complex catalogs.
2. The BioR indexer command currently does not tolerate duplicate keys, so while duplicate keys can be in the data itself, you can't index on those keys. Running `bior_index_catalog` with logging enabled will help to ensure the keys you would like to index on are valid. To index multiple ways simultaneously, multiple catalogs need to be created
3. Regardless of what tools are used to construct the JSON column, it must validate as proper JSON. Use `jslint` to validate: <http://jsonlint.com/>
4. JSON should not contain fields that are empty. While adding period "." As the value for a given key will work, it wastes space and consumes additional CPU resources so is not recommended.

## Use BioR to map SNP on rsID and find overlapping genes.

Say we obtained a simple tab-delimited file that is not in VCF format, but we still want to obtain an annotation. The following file's header for this is: `rsid` without the "rs", `chrom`, `position`, and `0/1` representing presence or absence in our study. There are over 5 million in this file. The goal is to show how the first 100 or 1000 of these map to various genes

```
$ zcat b132_SNPChrPosOnRef_37_1.bcp.gz | more
```

```
3      13      32446841      0
4      13      32447221      0
5       7      91839109      1
6       7      91747130      1
7       7      91779556      1
8       7      92408328      0
9       7      92373453      0
10      7      92383887      0
11      7      11364200      0
12      7      11337163      0
13      7      11387690      0
14      7      11380841      0
15      7      11602931      1
16      7      11602898      1
17      7      11583798      1
18      7      11597474      1
19      7      11597155      1
20      7      11597104      1
21      7      11596933      1
22      7      11596501      1
```

```
...
```

Try playing around with something like this to get started: (it may not be exactly what you want but we can work on that)

NCBIGene:

```

1  CHROM      1
2  POS        215848808
3  ID         rs116645811
4  REF        G
5  ALT        A
6  QUAL       .
7  FILTER     .
8  INFO       .
9  VCF2VariantPipe {
    "CHROM": "1",
    "POS": "215848808",
    "ID": "rs116645811",
    "REF": "G",
    "ALT": "A",
    "QUAL": ".",
    "FILTER": ".",
    "INFO": {
      ".": true
    },
    "_id": "rs116645811",
    "_type": "variant",
    "_landmark": "1",
    "_refAllele": "G",
    "_altAlleles": [
      "A"
    ],
    "_minBP": 215848808,
    "_maxBP": 215848808
  }
10 OverlapPipe {
    "_type": "gene",
    "_landmark": "1",
    "_strand": "-",
    "_minBP": 215796236,
    "_maxBP": 216596738,
    "gene": "USH2A",
    "gene_synonym": "dJ1111A8.1; RP39; US2; USH2",
    "note": "Usher syndrome 2A (autosomal recessive,
mild); Derived by automated computational analysis using gene prediction
method: BestRefseq.",
    "GeneID": "7399",
    "HGNC": "12601",

```



```

$ cat example.vcf | bior_vcf_to_tjson | bior_overlap -d $bior/NCBIGene/GRCh37_p10/genes.tsv.bgz | bior_drill -p Gene
| cut -f 9 --complement
##fileformat=VCFv4.0
#CHROM          POS          ID          REF          ALT          QUAL          FILTER        INFO          gene          GeneID
1              215848808          rs116645811      G            A            .            .            .            USH2A      7399
...
21              26965148          rs1135638        G            A            .            .            .            MRPL39    54148
21              26965172          rs010576         T            C            .            .            .            MRPL39    54148
21              26965205          rs1057885        T            C            .            .            .            MRPL39    54148
21              26976144          rs116331755      A            G            .            .            .            MRPL39    54148
21              26976222          rs7278168        C            T            .            .            .            MRPL39    54148
21              26976237          rs7278284        C            T            .            .            .            MRPL39    54148
21              26978790          rs75377686       T            C            .            .            .            MRPL39    54148
21              26978950          rs3989369        A            G            .            .            .            MRPL39    54148
21              26979752          rs61735760       C            T            .            .            .            MRPL39    54148
21              34022588          rs115683257      C            A            .            .            .            SYNJ1     8867
21              34029195          rs114053718      A            G            .            .            .            SYNJ1     8867
21              34058146          rs114942253      C            T            .            .            .            SYNJ1     8867
21              34059352          rs2254562        T            C            .            .            .            SYNJ1     8867
...
$

```

Now, we want to find "Approved\_Symbol", "Entrez\_Gene\_ID", "Ensembl\_Gene\_ID", "UniProt\_ID", ...  
We can use the BioR lookup command:

First, we don't know the catalog Structure of HGNC, here is a way to look at the structure of a catalog:

### Case Study: Creating a Report that Maps rsIDs to Genes.





```

],
    "Pubmed_IDs": [
        "2591067"
    ],
    "RefSeq_IDs": [
        "NM_130786"
    ],
    "Record_Type": "Standard",
    "Primary_IDs": [],
    "Secondary_IDs": [],
    "CCDS_IDs": [
        "CCDS12976.1"
    ],
    "VEGA_IDs": [],
    "mapped_GDB_ID": "GDB:119638",
    "mapped_Entrez_Gene_ID": "1",
    "mapped_OMIM_ID": "138670",
    "mapped_RefSeq": "NM_130786",
    "UniProt_ID": "P04217",
    "mapped_Ensembl_ID": "ENSG00000121410",
    "UCSC_ID": "uc002qsd.4",
    "mapped_Mouse_Genome_Database_ID": "MGI:2152878",
    "mapped_Rat_Genome_Database_ID": "RGD:69417"
}
$

```

To join the information in this catalog, to the information that we have collected in the gene table, we need to tell bior what field in the HGNC table matches the LAST column in our sample data + annotation. In this case, we will join on approved symbol (note: if you ever get an error with doing a lookup, you may need an index file - look into the `bior_index_catalog` command or contact the bior team for help).

```

grep "^22.*rs3721" gene_snp.dbl32.gene.coding.dat | more
22 7332 UBE2L3 rs372150 29047
22 150223 YDJC rs372150 23030
22 164592 CCDC116 rs372150 15754
22 23753 SDF2L1 rs372150 8782
22 23753 SDF2L1 rs372108 45008
22 23759 PPIL2 rs372150 -12903
22 23759 PPIL2 rs372108 0
22 29799 YPEL1 rs372150 -44455
22 29799 YPEL1 rs372108 -8229
22 83746 L3MBTL2 rs3721 0
22 150356 CHADL rs3721 0
22 5905 RANGAP1 rs3721 -14542

```



## 12. Sun Grid Engine

This section gives tips on how to configure a Sun Grid Engine (SGE) job to request the right amount of resources to successfully execute one or more BioR toolkit commands.

### Enable SGE at Mayo

At Mayo, for example, you can log onto an RCF system, such as crick7, then run “mayobiotools” and choose “69. ogs”, then select the available option, choose “0” to save and exit, then log out and back in again. You should now be able to run SGE commands such as “qsub”.

### Multiple Cores

By default, an SGE job will run on a single core. It’s possible to run a job on multiple cores is specified via the qsub command’s parallel environment option “-pe”.

```
-pe parallel_environment n[-[m]]|[-]m,...
```

To get a list of available parallel environments setup by your SGE admin:

```
> qconf -spl  
  
fluent_pe  
make  
mpich2_141_hydra  
mpich2_mpd  
namd2  
openmpi  
pvm  
pvm-tight  
threaded
```

Here is an example of requesting 4 cores for a job:

```
> qsub -pe threaded 4
```

The following table gives recommend core values for toolkit commands.

Command	Cores	Notes
Arbitrary UNIX commands	0	examples: /bin/cat, /bin/grep, /bin/cut
bior_vcf_to_tjson	1	
bior_overlap	1	
bior_same_variant	1	
bior_lookup	1	
bior_drill	1	

bior_compress	1	
bior_vep	2	<b>Warning:</b> Variant Effect Predictor is implemented using PERL. The virtual memory for the PERL process grows linearly with more variants.
bior_snpeff	2	SnEff loads data into memory for performance
bior_annotate	29	Annotate performs many commands in parallel
bior_pretty_print	1	

## Virtual Memory

Virtual memory is specified via the `qsub` command's resource request list option `"-l"`.

```
-l resource=value,...
```

NOTE: Resources specified with this option are **per-core**. If your job uses 2 cores, you will need to divide the resource value by 2.

For virtual memory, the resource name to use is `h_vmem`. Here is an example of requesting 10MB of virtual memory for a job running on 1 core:

```
> qsub -l h_vmem=10M
```

The following table gives recommend virtual memory values for toolkit commands.

Command	Virtual Memory	Notes
Arbitrary UNIX commands	100M	examples: /bin/cat, /bin/grep, /bin/cut
bior_vcf_to_tjson	600M	
bior_overlap	600M	
bior_same_variant	600M	
bior_lookup	600M	
bior_drill	600M	
bior_compress	600M	
bior_vep	1200M*	<b>Warning:</b> Variant Effect Predictor is implemented using PERL. The virtual memory for the PERL process grows

		linearly with more variants.
bior_snpeff	5100M	SnEff loads data into memory for performance
bior_annotate	24000M	
bior_pretty_print	225M	

## Resources for a Toolkit Pipeline

This section describes how to request the right resources for a multi-command Toolkit pipeline. Here is an example script that will be submitted to SGE:

```
> cat example.sh

#!/bin/sh

# dbSNP 137 catalog
DBSNP_CATALOG=/path/to/catalogs/dbSNP/137/00-All_GRCh37.tsv.bgz

# run toolkit pipeline to annotate my variants with dbSNP rsIDs
cat data.vcf | bior_vcf_to_tjson | bior_same_variant -d $DBSNP_CATALOG | bior_drill -p INFO.ID
```

The number of cores needed to run this script's processes in parallel can be calculated by referencing the table in the Multiple Cores section. The example script will require 3 cores to run optimally.

Command	Cores
example.sh	0
/bin/cat	0
bior_vcf_to_tjson	1
bior_same_variant	1
bior_drill	1

The virtual memory needed to run this script can be calculated by referencing the table in the Virtual Memory section. The example script will require 2000M of virtual memory (100 + 100 + 600 + 600 + 600).

Command	Virtual Memory
example.sh	100M
/bin/cat	100M
bior_vcf_to_tjson	600M
bior_same_variant	600M

bior_drill	600M
------------	------

The virtual memory setting `h_vmem` is specified on a **per-core** basis. Since `example.sh` will be using 3 cores and 2000MB of virtual memory total, `h_vmem` is  $2000/3$  or roughly 670.

## Find an Open Queue

```
> qconf -sql
```

Then choose a queue from the list to run your script under. For example, we'll assume there is a queue in the list called "MY\_QUEUE" which we'll use in the final command.

Here is the final `qsub` command with the correct resource requirements:

```
> qsub -m bae -M myemail@company.com -q 1-day -l h_vmem=5000M -pe threaded 3 -V -cwd example.sh
```

- cwd The -cwd param will specify output to go into the current directory (execute job from current dir).
- wd You can specify a target directory for output using -wd <pathToDir>.
- V Using -V param will export ALL of your environment variables.
- M Send email
- m Notify me by mail (with -M flag) when certain conditions occur

## Status of your Command

Find the status of your command - here it is waiting in the queue, but has not yet started processing:

```
> qstat
job-ID prior name user state submit/start at queue slots ja-task-ID
-----
119477 0.00000 example.sh m054457 qw 11/22/2013 09:28:11 3
```

After kicking off the process, it looks like:

```
$ qstat
job-ID prior name user state submit/start at queue slots
ja-task-ID
-----
119477 0.51062 example.sh m054457 r 11/22/2013 10:47:47 1-day@dnnode91.mayo.edu 3
```

## Get Command Results

The grid will output several files that begin with the script name you executed, and end with the queue jobId.

```
> ls -la example.sh.*
-rw-r--r-- 1 m054457 biostat 0 Nov 22 09:38 example.sh.pe119477
-rw-r--r-- 1 m054457 biostat 0 Nov 22 09:38 example.sh.pe119477
-rw-r--r-- 1 m054457 biostat 0 Nov 22 09:38 example.sh.o119477
-rw-r--r-- 1 m054457 biostat 283 Nov 22 09:38 example.sh.e119477
```