

Advanced breseq GVA2022

- [Overview](#)
- [Learning Objectives](#)
- [Get some data](#)
- [Setting up the run](#)
- [Breseq commands](#)
 - [Commands file](#)
 - [Slurm file](#)
 - [Evaluating the run](#)
- [Using gdttools to compare samples.](#)
 - [Grouping .gd files together](#)
 - [Generating a compare table](#)
- [Evaluating compare table](#)
- [Exporting multiple breseq results using a single .tar.gz file](#)
- [Inflating compressed archives of breseq results.](#)
- [Comparing breseq output to compare table](#)
- [Other useful tutorials](#)

Overview

We analyzed a single [lambda phage evolution experiment](#) as both an introduction to variant visualization in the first part of Wednesday's class, but *breseq* is designed to be able to with "small" genomes, not just "tiny" genomes. It also has more advanced tools for visualizing mutations across multiple genomes and more complete identification of all mutations.

Learning Objectives

1. Run breseq on 7 different E. coli clones evolved for 40,000 generations
2. Use the packaged gdttools commands to generate a comparison table of all mutations from all 7 samples
3. Learn shortcuts for compressing, transferring, and extracting multiple folders/files with single commands
4. Visualize the data on your local computer

Get some data

The data files for this example are in the following path: **\$BI/ngs_course/ecoli_clones/data/** go ahead and copy them to a new folder in your \$SCRATCH directory called **GVA_breseq_multi-sample**:

Suggested command

```
mkdir $SCRATCH/GVA_breseq_multi-sample
cd $SCRATCH/GVA_breseq_multi-sample
cp $BI/ngs_course/ecoli_clones/data/* .
```

if everything worked correctly, you should see the following files. We've provided a bit more context to what those files actually are:

File Name	Description	Sample
SRR030252_1.fastq SRR030252_2.fastq	Paired-end Illumina 36-bp reads	0K generation evolved E. coli strain
SRR030253_1.fastq SRR030253_2.fastq	Paired-end Illumina 36-bp reads	2K generation evolved E. coli strain
SRR030254_1.fastq SRR030254_2.fastq	Paired-end Illumina 36-bp reads	5K generation evolved E. coli strain
SRR030255_1.fastq SRR030255_2.fastq	Paired-end Illumina 36-bp reads	10K generation evolved E. coli strain
SRR030256_1.fastq SRR030256_2.fastq	Paired-end Illumina 36-bp reads	15K generation evolved E. coli strain
SRR030257_1.fastq SRR030257_2.fastq	Paired-end Illumina 36-bp reads	20K generation evolved E. coli strain
SRR030258_1.fastq SRR030258_2.fastq	Paired-end Illumina 36-bp reads	40K generation evolved E. coli strain


NC_012967.1.fasta NC_012967.1.gbk	Reference Genome	<i>E. coli</i> B str. REL606
-----------------------------------	------------------	------------------------------

Setting up the run


As we begin working with multiple samples it becomes more important or at least helpful to organize our output in a more meaningful way. Typically this means storing input files in 1 location, output folders in another, and information about the analysis in a 3rd location.

Suggested command

mkdir run_output logs

 Note in the above command we are making deliberate use of having a space on the command line to create 2 different directories as was mentioned in the introduction tutorial.

On stampede2 with both R1 and R2, the run takes ~5 hours to complete making it a task for the job queue system rather than an idev node. When we begin to make our slurm file, we will request 8 hours to be on the safe side (this behavior will be discussed on Friday). One way to speed up such analysis is to limit the amount of data being analyzed such as only looking at R1 or R2 instead of both.

 **Why might "only" looking at R1 data be ok?**

Throwing out half your data probably seems like a bad idea, certainly a waste of money. Sometimes, it does not impact your actual analysis, and sometimes it can actually improve your analysis.

1. If you have more coverage than you need (ie more than ~100x coverage of a clonal sample), downsampling your data to use fewer reads can improve both the analysis speed and accuracy as fewer errors will be present. **If the analysis pipeline you are using does not make use of paired end information**, this can be accomplished by only looking at R1, if not, you could use the head or tail commands to extract a given number of reads. Some programs also include options to limit coverage to a certain level and go about this in different ways but require no modification of your read files. breseq is both capable of limiting coverage and also does not use paired end information making using only R1 a reasonable strategy
2. While rare, it is possible that R1 or R2 simply has much lower quality scores than the other read. In this case it would be appropriate to drop 1 read or the other from your analysis. This would be something you could determine using fastqc and/or multiqc which have their own [tutorial at MultiQC - fastQC summary tool -- GVA2021](#)

Breseq commands

As you saw in the initial [Breseq Tutorial](#) and possibly other tutorials already, using the job queue system requires 2 things, a commands file, and a slurm file to control the remote computer.

Our generic breseq command will be the following:

Generic command

breseq -j 9 -r NC_012967.1.gbk -o run_output/<XX>K <ID>_1.fastq <ID>_2.fastq &> logs/<XX>K.log.txt

And its parts can be explained as follows:

part	purpose
-j 9	use 9 processors/threads speeding things up by less than a factor of 9 as discussed in earlier tutorials
-o run_output/<xx>k	directs all output to the run_output directory, AND creates a new directory with 2 digits (<XX>) followed by a K for individual samples data. If we don't include the second part referencing the individual sample, breseq would write the output from all of the runs on top of one other. The program will undoubtedly get confused, possibly crash, but definitely not be analyzable
<ID>	this is just used to denote read1 and or read2 ... note that in our acutal commands they reference the fastq files, and are supplied without an option

<pre>&> logs/<X X>00K. log.txt</pre>	<p>Redirect both the standard output and the standard error streams to a file called <XX>00k.log.txt. and put that file in a directory named logs. The &> are telling the command line to send the streams to that file.</p>
---	---



Why do we use -j 9?

1. Each stamped2 compute node has 68 processors available.
2. We have 7 samples to run, so by requesting 9 processors, we allow all 7 samples to start at the same time leaving 5 unused processors.
3. If we had requested 10 processors for each sample, only 6 samples would start initially and the 7th would start after the first finishes.

Commands file

Enter the lines below into a new file named "breseq_commands" using nano.

This is the expected nano command, not the text that goes in the file.

```
nano breseq_commands
```

```
breseq -j 9 -r NC_012967.1.gbk -o run_output/00K SRR030252_1.fastq SRR030252_2.fastq &> logs/00K.log.txt
breseq -j 9 -r NC_012967.1.gbk -o run_output/02K SRR030253_1.fastq SRR030253_2.fastq &> logs/02K.log.txt
breseq -j 9 -r NC_012967.1.gbk -o run_output/05K SRR030254_1.fastq SRR030254_2.fastq &> logs/05K.log.txt
breseq -j 9 -r NC_012967.1.gbk -o run_output/10K SRR030255_1.fastq SRR030255_2.fastq &> logs/10K.log.txt
breseq -j 9 -r NC_012967.1.gbk -o run_output/15K SRR030256_1.fastq SRR030256_2.fastq &> logs/15K.log.txt
breseq -j 9 -r NC_012967.1.gbk -o run_output/20K SRR030257_1.fastq SRR030257_2.fastq &> logs/20K.log.txt
breseq -j 9 -r NC_012967.1.gbk -o run_output/40K SRR030258_1.fastq SRR030258_2.fastq &> logs/40K.log.txt
```

Be sure to write (**ctrl + o**) before you exit (**ctrl +x**) nano.

Slurm file

Modify your slurm file

```
cp /corral-repl/utexas/BioITeam/gva_course/GVA2022.launcher.slurm breseq.slurm
nano breseq.slurm
```

Again while in nano you will edit most of the same lines you edited in the in the [breseq tutorial](#). Note that most of these lines have additional text to the right of the line. This commented text is present to help remind you what goes on each line, leaving it alone will not hurt anything, removing it may make it more difficult for you to remember what the purpose of the line is

Line number	As is	To be
16	#SBATCH -J jobName	#SBATCH -J mutli_sample_breseq
17	#SBATCH -n 1	#SBATCH -n 7
21	#SBATCH -t 12:00:00	#SBATCH -t 6:00:00
22	##SBATCH --mail-user=ADD	#SBATCH --mail-user=<YourEmailAddress>
23	##SBATCH --mail-type=all	#SBATCH --mail-type=all
29	export LAUNCHER_JOB_FILE=commands	export LAUNCHER_JOB_FILE=breseq_commands

The changes to lines 22 and 23 are optional but will give you an idea of what types of email you could expect from TACC if you choose to use these options. Just be sure to pay attention to these 2 lines starting with a single # symbol after editing them.

Again use **ctl-o** and **ctl-x** to save the file and exit.

You should verify that you will be attempting to access a conda environment on line 27 above that has access to breseq.

```
conda activate GVA-breseq
breseq --version
```

assuming you verify access to breseq version 0.36.1 you can then submit the job:

submit the job to run on the que

```
sbatch breseq.slurm
```

Evaluating the run

Generic information about checking the status of a run in the job queue system can be found in our earlier tutorial: [Stampede2 Breseq Tutorial GVA2022#Checkingyoursubmittedjob](#), but that has to do with TACC running the command, not with breseq finishing the analysis and how to verify it completed successfully. As noted above, our commands included "&>" followed by a file name so that all of the information that would normally print to the screen, instead printed to a log file. You can use the less command to evaluate any of the log files to see the information that would have been printed to the screen if we had run the commands interactively on an idev node. As you may have noticed from the earlier breseq runs, the final line of the breseq run is "+++ SUCCESSFULLY COMPLETED" when things have finished without errors. Since we have 7 samples, it is useful to check for this all at once:

You can also check how many have finished with this command

```
tail -n 1 logs/*|grep -c "+++    SUCCESSFULLY COMPLETED"
```

The above will return "7" if everything has gone correctly.



One of the most common commands I use in evaluating breseq runs completion status

While it is easy to keep track of only having 7 samples, jumping around between projects and collaborations it can sometimes be difficult to remember what number of samples i included on any given run. Further, when you start having dozens-100s of samples, with unknown read counts and reference genomes of varying quality, my time estimate of how long you need the job to run for can be incorrect. This one liner is useful for listing both the number of samples, and the number of samples that finished correctly. If there is a discrepancy I know I need to dig in deeper to address an issue, if they agree I can move on with the analysis.

```
wc -l *commands; tail -n 1 logs/*|grep -c "+++    SUCCESSFULLY COMPLETED"
```

I use *commands as I sometimes have different names for my commands file depending on what project I am working on, but this check always should work.

Using gdttools to compare samples.

Either after your samples are done running, or for those reading ahead, the next part of our analysis is done using **gdttools** commands. **gdttools** is a series of commands designed to function on "genome diff" (.gd) files (which are the main data type produced by breseq which is used by downstream analysis) that come packaged with breseq (so it doesn't require a separate conda installation). As we have seen with numerous other programs throughout the course typing **gdttools** by itself will return a list of options. In the case of gdttools, it returns the list of subcommands that are available. Each has its own use, but for the purpose of this tutorial we are interested in comparing the output of multiple samples to one another.

from the gdttools output we see that:

```
ANNOTATE (or COMPARE)  annotate the effects of mutations and compare multiple samples
```

Since **gdttools compare** seems to be the command and subcommand that we wanted to run to generate the comparison. Try typing just that to get a list of expected arguments.

The example command shows:

```
gdtools ANNOTATE/COMPARE [-o annotated.html] -r reference.gbk input.1.gd [input.2.gd ... ]
```

note that things displayed in the [] are optional.

In our command we sent the output of each sample to its own folder using the **-o run_output/<XX>K** portion of the command line. Among the multiple directories that breseq created within each output directory, you will find one named output which contains a file named **output.gd**. These are the files that we need to feed into the **gdtools compare** command (along with the **NC_012967.1.gbk** reference file) to generate the comparison table we want. In this course we have mentioned that using consistent nomenclature is important which it is, but now we come to a point where the consistency has created a problem. Because all of the files are named "output.gd" we can neither move all of the commands into a single folder (they would overwrite one another as they have the same name) or the resulting table would display sample IDs as "output" "output" "output" ... not the most helpful information to say the least.

To circumvent that we will need to rename each file to something more meaningful (say the name of the directory that the output/output.gd file was found in (aka the sample name)).



The remaining portion of this tutorial requires the runs to have completed

You can also check how many have finished with this command

```
cd $SCRATCH/GVA_breseq_multi-sample
tail -n 1 logs/*|grep -c "+++    SUCCESSFULLY COMPLETED"
```

Until the above returns 7 the following commands **will not work correctly**. If your job is no longer listed in the queue system (use "showq -u" to check), and you do not get "7" back as the result let me know and we'll troubleshoot the issue together.

Grouping .gd files together

Handy command to move into the run_output directory, and copy all those .gd file to a new folder with a new name

```
cd $SCRATCH/GVA_breseq_multi-sample/run_output
for d in *;do cp -i $d/output/output.gd $d.gd;done;mkdir gd_files;mv -i *.gd gd_files
mv gd_files ..
```

This will copy the .gd file from each run into a new directory and then move that directory up 1 level so that it will be easier to use when making the compare table

Generating a compare table

From the information above we can construct the following command to generate the compare table

```
cd $SCRATCH/GVA_breseq_multi-sample/
gdtools compare -r NC_012967.1.gbk -o multi-sample-compare-table.html gd_files/*.gd
```

where:

- -r ... is the reference
- -o ... is the output table we want
- gd_files/*.gd lists all the samples we want to have on our table

As the .html ending suggests, it will be an html output similar to the other breseq output you have seen and thus needs a web browser to be able to view it. Therefore you should use the scp command to transfer the multi-sample-compare-table.html file back to your local computer. [Remember assistance with the scp command can be found here.](#)

Evaluating compare table

Once you open the file on your local computer consider the following broad explanations:

1. Each row represents a unique mutation
2. The first 2 columns state what the position in the genome is, and what mutation occurred
3. Columns 3-9 represent each individual sample

4. the remainder of the columns provide information from the annotation describing what genes are hit, and what the effect of the mutation is

As columns 3-9 show an increase in evolutionary time (from 2,000 to 20,000 generations), you will notice multiple examples where the first few columns are empty but the last 2,3,4 columns show 100% showing that the mutation was not detected until a certain time point then arose before the next sampling, and remained in the population to the end of our time course.

Can you find any examples of mutations that arose, but were out competed by the end of the evolution experiment? (do you understand what those should look like)

One downside of compare tables is that while it shows what mutations breseq called, you lose the ability to click on individual mutations to understand how /why that mutation was called. Therefore while compare tables are great, they are nearly always used in conjuncture with evaluating the individual breseq output as well. The next section will deal with a helper script to more quickly export multiple samples.

Exporting multiple breseq results using a single .tar.gz file

I have a small shell script for dealing with breseq data that copies, renames, compresses, groups each samples output directory into a single new directory, and compresses that directory so you can easily transfer it. Where can you get ahold of such a time saving script you ask? In the BiolTeam directories of course!

Using Dan's most commonly used shell script for breseq data

```
cd $SCRATCH/GVA_breseq_multi-sample/run_output
ls
ls ..
export-breseq.sh
ls
ls ..
```

The above code block:

1. moves to the directory that contains the samples we just ran through breseq
2. shows you what is in that directory
3. shows you what is in the parent directory
4. executes the bash script described above
5. shows you the new output is in the parent directory not the current directory

Using the [SCP tutorial if necessary](#) transfer the compressed archive 05_Output_Export.tar.gz back to your computer.

Inflating compressed archives of breseq results.

Command to type in the local terminal window AFTER the scp command

```
# scp command ... 05_Output_Export.tar.gz .
tar -xvzf 05_Output_Export.tar.gz
cd 05_Output_Export
ls
```

Now you will see 7 .tar.gz files in the directory, and you can extract each of them 1 at a time by using the **tar -xvzf <FILE>** command. This seems acceptable for 7 samples but not when dealing with 100s of samples. The 1 liner below will extract .tar.gz files sequentially:

```
for f in *.tar.gz;do tar xvzf $f;done
```

This command will extract all files ending in .tar.gz regardless of their origin. If you find yourself doing this routinely you may want to turn this command into a small bash script as i have done. This is as simple as the following 2 commands and moving the detar.sh file to a location in your \$PATH. Remember you can check what directories are in your \$PATH by typing **echo \$PATH**

```
echo "for f in *.tar.gz;do tar xvzf $f;done" > detar.sh
chmod +x detar.sh
```

This command will work on either your personal computer or TACC the same way, but in my experience, the export-breseq.sh is used on TACC while detar.sh is used on my local computer.

Comparing breseq output to compare table

Now you can click through each individual sample's output to see the different mutations as you could in the [intro to breseq tutorial](#).

Other useful tutorials

Data analysis with breseq (like all other programs) is only as good as the data that goes into it. The MultiQC and fasp tutorials work well upstream of breseq, while the identification of novel DNA elements may be useful for things such as trying to identify unknown plasmids and makes use of the genome assembly tutorial.

Alternatively, again consider going back to the [intro breseq tutorial](#) to rerun the sample that you took through [read QC](#), [mapping](#), [SNV calling](#), and [visualization](#).

[Back to the GVA2022page](#)