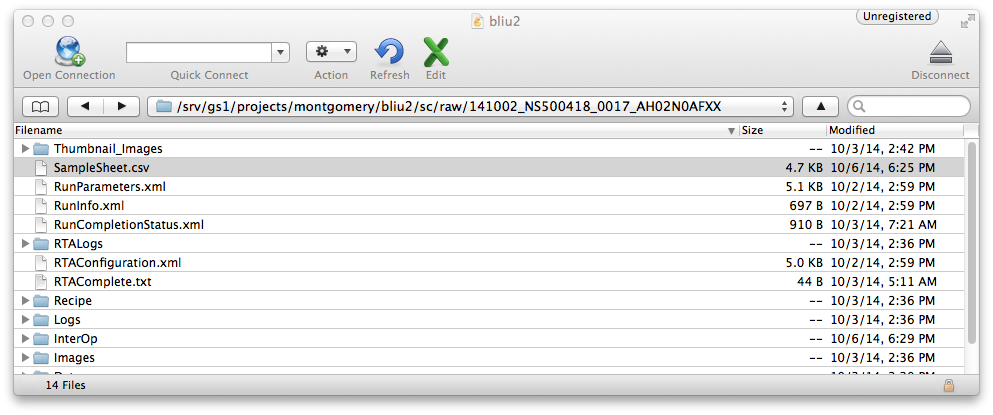
# TOOLS

# bcl2fastq

First, put a “SampleSheet.csv” in the top directory. E.g.



Sample sheet templates:

* **NextSeq500\_SampleSheet\_DualIndex.csv:** template for dual index
* **NextSeq\_SampleSheet\_SingleIndex.csv:** template for single index

To convert NextSeq bcl files to fastq format, use **run\_bcl2fastq2.sh**

# BWA

read\_group\_info="@RG\tID:group1\tSM:sample1\tPL:illumina\tLB:lib1\tPU:unit1"

bwa mem -M -R ’<read group info>’ -p reference.fa raw\_reads.fq > aligned\_reads.sam

* Help: [here](https://www.broadinstitute.org/gatk/guide/tagged?tag=bwa).
* Run BWA over a directory: **run\_bwa\_single\_end.sh**

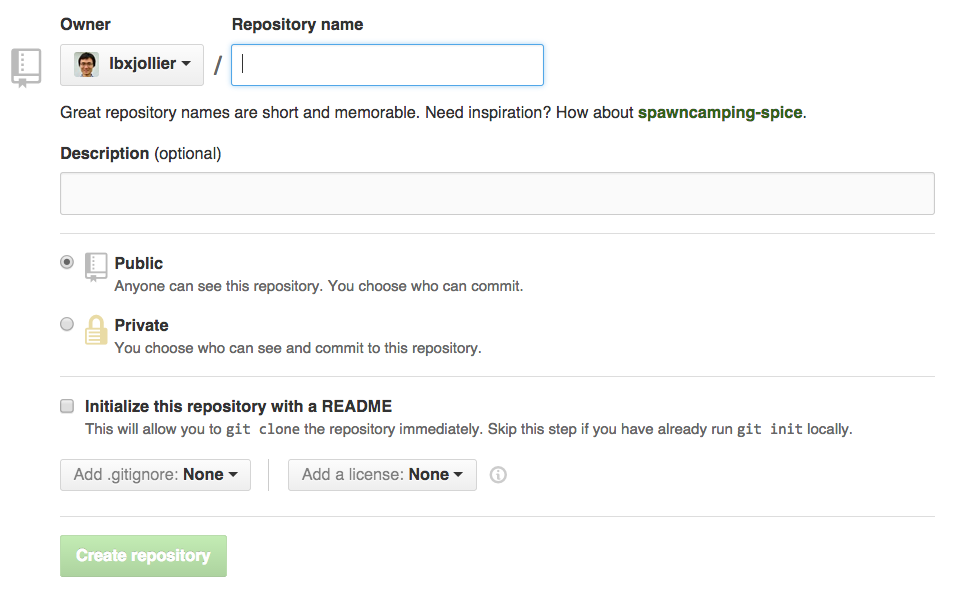
# GATK

# Git

## Basic operations

To be a successful bioinformatician, the first thing you need to do is to create a git repository to store your scripts.

First, let us create a git remote repo on github. If you don’t have a github account, you should really create one on <https://www.github.com>. After you have a personal account, clikc on the ‘plus’ button on the topright corner to create a new repository. You will see the following page.



Enter repository name, and click on ‘Create repository’.

Now clone this repository to your local machine:

*$ git clone git@github.com:lbxjollier/bioinformatics\_toolbox.git*

# If you encounter the following error, follow this [link](https://help.github.com/articles/generating-ssh-keys/) to generate a public ssh key.

*Permission denied (publickey).*

*fatal: The remote end hung up unexpectedly*

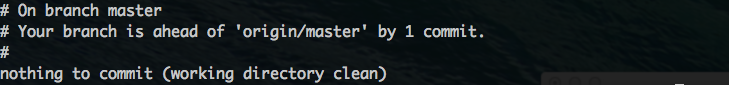
This command will create a directory named “bioinformatics\_toolbox”. Add some files into this directory, and type:

Let usdo our first commit:

*$ git commit –m ‘initial commit’*

This command tells git to keep a permanent copy of the current files. To check whether a desired files have been committed, use:

*$ git status*

You will see: 

This reflects that we have committed the newly added files locally, but have not ‘pushed’ the local file to our remove repo on github. To push, use:

*$ git push origin master*

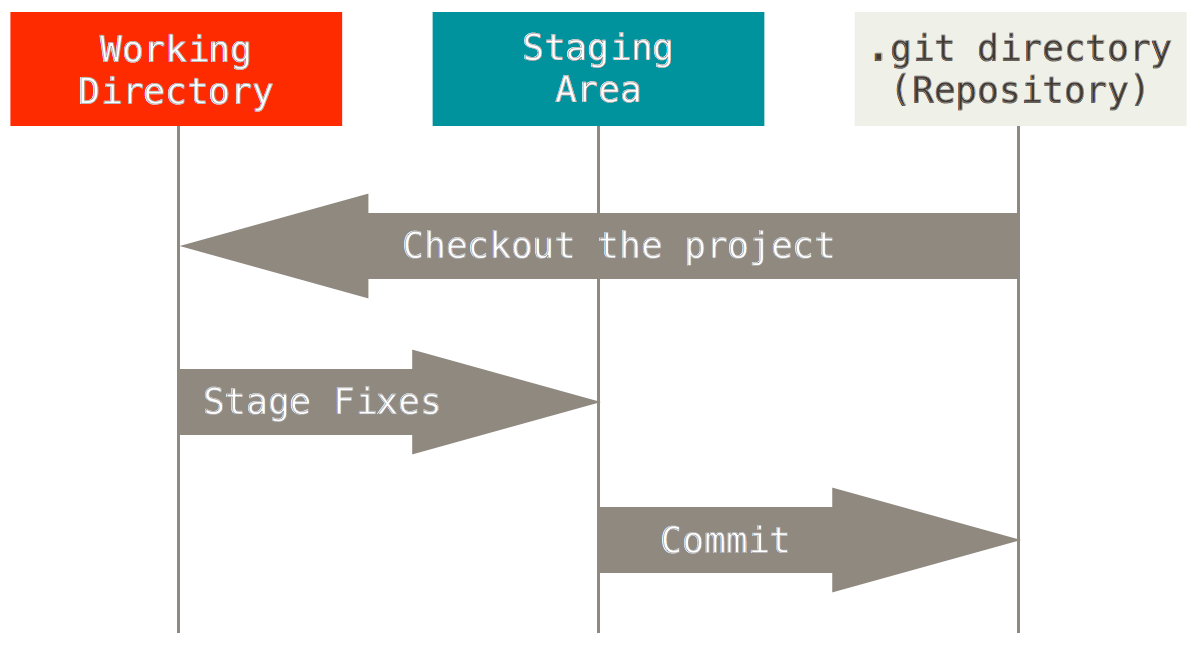
To exclude a file, first create a .gitignore file:

$ touch .gitignore

Then add files you don’t want to include. For example:

$ echo '.nfs0000000006ee455100000edb' >> .gitignore

## Interactive Staging



“Git has three main states that your files can reside in: committed, modified, and staged. Committed means that the data is safely stored in your local database. Modified means that you have changed the file but have not committed it to your database yet. Staged means that you have marked a modified file in its current version to go into your next commit snapshot.”

To interactively stage files, use:

$ git add -i

## More information:

* git-scm interactive staging: [link](http://git-scm.com/book/en/v2/Git-Tools-Interactive-Staging)
* minimal tutorial: [link](http://kbroman.org/github_tutorial/)

# HTSeq

HTSeq is a python package that provides tools manipulate sam files and fastq files. Below is a summary of some of its important functionalities.

## Sam Quality Control with HTSeq-qa

TBD

## Count Features with HTSeq-count

* --idattr=<gene\_name/gene\_Id>
* gene\_name: e.g. DDX11L1
* gene\_id: e.g. ENSG00000223972
* --format=<sam/bam>
* --stranded=<yes/no/reverse>

### More information

* HTSeq-count: [link](http://www-huber.embl.de/users/anders/HTSeq/doc/count.html)
* Ensemble GFF specs: [link](http://uswest.ensembl.org/info/website/upload/gff.html)
* Sanger GFF definition (more detailed): [link](http://www.sanger.ac.uk/resources/software/gff/spec.html#t_2)

# IGV

## Start IGV

java –Xmx1000m –jar path/to/igv.jar –b <file\_names> -g hg19

hg 19 tells igv to load hg19 (from Broad server). You can also specify a path to a local genome file.

# Picard

## Mark Duplicate

java -Xmx4g -jar MarkDuplicates.jar \

INPUT=sorted\_reads.bam \

OUTPUT=dedup\_reads.bam \

METRICS\_FILE=metrics.txt

* Batch implementation: **mark\_duplicates\_picard.sh**
* More information: GATK mark duplicates [tutorial](https://www.broadinstitute.org/gatk/guide/tagged?tag=markduplicates)

## Compress, Sort and Index

Picard's SortSam function is more stringent than samtools. It will report error when there is an empty line.

java -Xmx2g -jar $picard/SortSam.jar INPUT=$input\_dir/${inputs[$i]} OUTPUT=/dev/stdout SORT\_ORDER=coordinate | \

java -Xmx2g -jar $picard/SamFormatConverter.jar INPUT=/dev/stdin OUTPUT=$output\_dir/$output CREATE\_INDEX=true

* Notice you can pipe using /dev/stdout and /dev/stdin
* Over a directory : compress\_sort\_and\_index\_picard.sh

# Python

## DIY Python Module

Refer to this [page](http://www.ibiblio.org/g2swap/byteofpython/read/making-modules.html).

## Enumerate

## Regular Expression

## Catching Bugs with "Assert"

Look [here](https://wiki.python.org/moin/UsingAssertionsEffectively).

# R

## data.table

cheat sheet: [here.](https://s3.amazonaws.com/assets.datacamp.com/img/blog/data+table+cheat+sheet.pdf)

Comparison between data.table and dplyr: [here](https://stackoverflow.com/questions/21435339/data-table-vs-dplyr-can-one-do-something-well-the-other-cant-or-does-poorly)

## Data Types

* vectors: 1D collection of values of the **same** type.
* matrices: 2D collection of values of the **same** type.
* arrays: 3+ dimension collection of values of the **same** type.
* data.frame: 2D collection of values, allowing **different** types.
* lists: 1D collection of **different objects**, be it data.frames, lists, etc..
* factors: 1D collection of **nomial** values.
* more info: [here](http://www.statmethods.net/input/datatypes.html).

## dplyr

* filter(): select rows by condition
* slice(): select rows by index
* select(): subset columns
* rename(): rename columns
* arrange(): order rows by value
* mutate(): create new columns that are results of operations on existing columns.
* transmute(): similar to mutate() but only returns new columns
* summarise(): as name suggests..
* sample\_n(): sample n items
* sample\_frac(): sample a fraction
* group\_by(): as name suggests…

Get started: [here](http://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html)

## distributional functions

* normal distribution: rnorm(), dnorm(), pnorm(), qnorm()
* factors: gl()
* …

## Exception Handling

### Assertions

assertthat package from Hadley.

### Try-Catch

TBD.

Nice [blog](http://mazamascience.com/WorkingWithData/?p=912)

## How To Make An R Package

You can make your own R package! It can be done under an hour using the instructions here: [link](http://hilaryparker.com/2013/04/03/personal-r-packages/)

If you’d like to delve deeper into making your package, Hadley Wickham wrote a [book](http://r-pkgs.had.co.nz/intro.html) on this subject.

In this tutorial, I am quoting the bare minimum of the essential components from creating and documenting your package, to pushing your package onto Github. In addition, I am teaching by way of examples. If you would like your own package just switch out some variables in the examples.

### Setup

Install required packages:

install.packages(c("devtools", "roxygen2", "testthat", "knitr"))

Creating an R package called brt under "~/brt":

devtools::create('~/brt')

A few shortcuts to memorize:

* Cmd + Shift + B: build and reload
* Cmd + Shift + L: load all functions

After modifying a function, use these two shortcuts to renew your package content.

To navigate to a function, you can either:

* Ctrl + . and search
* Click on a function name and press F2

Library and package are two different concepts. A package is a bundle of R scripts. A package is a directory with packages.

### Dependencies

To import a package, modify the import line in the DESCRIPTION file (start a import line if there isn't already one). Say I want to import the package 'stringr' and ggplot2, I would:

Imports: stringr (>= 0.6.2), ggplot2

The parenthesis following stringr specifies the minimum version.

Everytime your package is attached, the packages you listed under imports will be **installed but not attached**. In other words, you need to specify the package namespace using :: to call functions under that package. E.g.:

stringr::str\_replace()

This is the recommended practice as it clearly shows which package the function is called from. However, if a function is repeatly called, you can choose to import its namespace in the file NAMESPACE. E.g.:

import(stringr)

After this import, you can call functions without referring to their package names anymore.

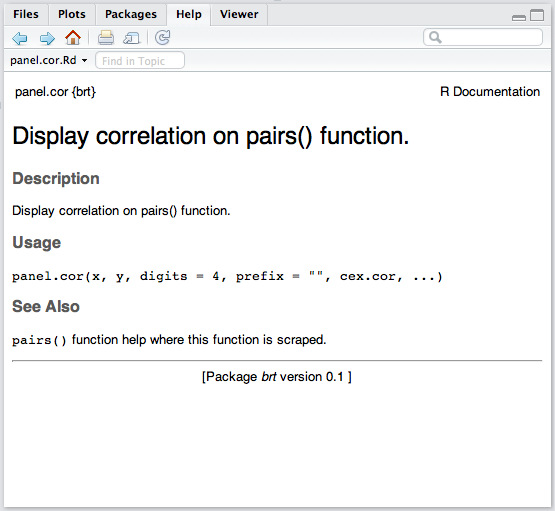
### Documenting Functions

By way of example, if I want to document my function panel.cor(), I would add the following lines preceding the function definition.

#' Display correlation on pairs() function.

#'

#' @seealso \code{pairs()} function help where this function is scraped.

****

The corresponding help page is the follows. You can also put @examples, @author lines, etc.

Once you finish adding documentation lines, type:

devtools::document()

This command generate a <function>.Rd file in the man/ directory.

## Version Control

see [here](http://r-pkgs.had.co.nz/git.html).

## Heatmap

Use heatmap.2, below are some options:

* Rowv = FALSE, disable row dendrogram
* Colv = FALSE, disable column dendrogram
* trace = 'none', disable the green traces
* dendrogram = 'none', disable dendrograms.

## Read Data

**File with unknown number of fields:**

ncol = max(count.fields(filename, sep = '\t'))

col.classes = rep('NULL', ncol)

col.classes[c(5,9,13)] = c(rep(<class of the column>,3)) # only read column 5,9, and 13.

df = read.table(filename, colClasses = col.classes, fill = T, col.names = 1:ncol)

## R Markdown

* list –
* title -------
* italics \* \*
* bold \*\* \*\*
* in-line code ` `
* in-line equation $ $
* equation $$ $$
* block quotes >

Reference: [here](http://rmarkdown.rstudio.com/)

## R Presentation

You can make presentations with R.

Some options:

* title:{true,false}
* set global cache

```{r setup, include=FALSE}

opts\_chunk$set(cache=TRUE)

```

* add image

!["here is my caption"](myimage.png)

* Two column: \*\*\*
  + change size of one column:

left: 70%

* More: [here](https://support.rstudio.com/hc/en-us/articles/200486468-Authoring-R-Presentations?version=0.98.953&mode=desktop)

## R Project

R projects are isolated R environments, each with a separate .RHistory, .RData. .RProfile, etc. Each project also has an \*.Rproj file. Opening it will open the R project folder it resides in.

The \*.Rproj file is a simple text file with preferences for an R project.

## String Manipulation

I use stringr. It’s manual can be found [here](http://cran.r-project.org/web/packages/stringr/stringr.pdf).

## Style Guide

Writing beautiful code is not required by recommended. Hadley has a nice chapter in his R Package book: [here](http://r-pkgs.had.co.nz/style.html).

## Time

I recommend Hadley's **lubridate** package.

# Samtools

## Converting Bam Format to Pileup Format

samtools mpileup -B -f <genome.fa> <bam> -l <target\_sites.bed> > <output.pileup>

* -B Disable probabilistic realignment for the computation of base alignment quality (BAQ). BAQ is the Phred-scaled probability of a read base being misaligned. Applying this option greatly helps to reduce false SNPs caused by misalignments.
* -f [f]aidx indexed genome.fa file
* -l a [l]ist of target sites in bed format
* Run over a directory: **bam2pileup.sh**

# SCG3 Cluster

## qlogin

Some frequently used options:

* -pe shm <num of cores you need>
* -P large\_mem (to request large memory machine)
* -l h\_vmem=10G (request 10 G of memory)
* -l h\_vrt=96:00:00 (request 96 hr 0 min 0 seconds of run time)
* -N <name of project>

## X11 Forwarding

You can open applications like Rstudio and firefox on the remote cluster and forward the application screen through ssh. To enable X11 forwarding, type:

ssh <user>@scg3.stanford.edu –X

Taking advantage of X11 forwarding, you can use commands like:

* rstudio
* firefox
* gnome-open (to open pdf and other files)

# Shell Scripting

## Array

To covert string to array:

<array>=( $<string> )

To get length of an array:

${array[\*]}

To get element k from an array (0-based):

${array[k]}

## Conditionals

The if-elif-then statement. E.g.

if [ "$a" == "$b" ]; then

echo "a is b"

elif [ "$a" == "$c" ]; then

echo "a is c"

else

echo "a is neither b or c"

fi

Note:

* A space must exist after "[" and before "]"

Another quick and dirty way:

<condition> && ( statements to evaluate if condition is true )

<condition> || ( statements to evaluate if condition is false )

## Conditional Operators

**Integar Comparison**

* -ne not equal
* -eq equal

**String Comparison**

* == equal
* != not equal

More operators refer to this [page](http://tldp.org/LDP/abs/html/comparison-ops.html).

**File test operators**

* –f file exist
* -s file has content (not empty)

File test operators: [here](http://tldp.org/LDP/abs/html/fto.html)

## Command Line Options/Flags

Use **getopts.** Every time getopts is called, it move $OPTIND up 1 to access next command line option until the last the option and returns FALSE. A while loop can be used to sequentially parse every command option. Example:

while getopts ":i:o:" opt; do

case $opt in

i) input\_dir=$OPTARG

;;

o) output\_dir=$OPTARG

;;

\?) echo "Invalid option -$OPTARG"; exit 1

;;

esac

done

* OPTARG (a getopts keyword) stores the **argument** in current loop
* opt stores the **option/flag** in current loop
* ":i:o:" tells getopts to expect -i and –o. The colon after each letter tells getopts to expect an companion argument.

## Disk Quota

Use this command to check disk quota:

*ssh scg3-0-1 /srv/gs1/software/scg\_quota -p montgomery*

## Disk Usage

To check the size of a directory:

du –sh <directory>

To check the size of a directory and its sub-directories:

du –h <directory>

## Empty the content of a file

$ > $filename

Or more explicitly:

$ echo –n "" > $filename

* -n do not output the trailing newline character

## Grep

* -v reverse; find lines that do not contain a pattern.

## Find

The find command in one of the most versatile tools on Linux.

To find files larger than 1G under home directory.

*find ~ -type f –size +5G –exec ls –lh {} \;*

## Read File

while read line; then

echo "current line: $line"

done < $filename

## Translate (tr)

delete trailing newline charater

tr –d "\n"

## Variables

substitution

${var/<string\_to\_be\_replaced>/<replacement>}

To know more about variable manipulation, click [here](http://stackoverflow.com/questions/2188199/how-to-use-double-or-single-bracket-parentheses-curly-braces).

# STAR

## Generate STAR genome index

TBD

Two versions of STAR genome:

* genecode version 14

/srv/gs1/projects/montgomery/tnance/genomes/STAR/hg19\_gencode14\_overhang99

* gencode version 21

More information:

* sjdbOverhang clarification: [link](https://groups.google.com/forum/#!topic/rna-star/h9oh10UlvhI)

## STAR 1-pass

* Use **run\_star.sh**

Currently run\_star.sh only supports paired-end alignment. I will add single-end alignment in the future.

# SQLite

# PROTOCOLS

# Calling Variants using Samtools

## Alignment

In order to call variants, the first step is to align the short reads to the reference genome. In this case, we will be using Bowtie2 as our aligner. For a description of Bowtie2, click [here](http://bowtie-bio.sourceforge.net/bowtie2/index.shtml).

I am using Bowtie2 on the Stanford scg cluster, so I first login to a large memory node and load bowtie using:

$ ssh <user\_id>@greenie

$ module load bowtie/2.2.1

The basic command of bowtie2 is:

$ bowtie2 -x <bowtie2\_index> -U <input\_fastq\_files> -S <output\_sam\_files>

The –x, -U and –S are called flags. There meanings are:

* -x indicates this command line input is the bowtie2 inde[x].
* -U Comma-separated list of files containing [U]npaired reads to be aligned, e.g. lane1.fq,lane2.fq,lane3.fq,lane4.fq. Reads may be a mix of different lengths. If - is specified, bowtie2 gets the reads from the "standard in" or "stdin" filehandle.
* -S output [S]am files

To make our lives easier, let’s create a few atlases first:

fastq\_dir=<directory\_to\_fastq>

sam\_dir=<directory\_to\_sam>

index\_dir=<directory\_to\_bowtie\_index>

log\_dir=<directory\_to\_logs>

Now let us run bowtie2:

bowtie2 -x $index\_dir/<index\_basename> -U $fastq\_dir/<fastq\_file> -S $sam\_dir/<sam\_file> 2><log\_file>

The last part of the command “2> <log\_file>” redirects stderr to a log file for later reference.

A moment later, bowtie2 spits out:

137294 reads; of these:

137294 (100.00%) were unpaired; of these:

2189 (1.59%) aligned 0 times

128799 (93.81%) aligned exactly 1 time

6306 (4.59%) aligned >1 times

98.41% overall alignment rate

That’s pretty much all we need to do about alignment. Simple, right? A few additional resources.

* Bowtie2 manual ([link](http://bowtie-bio.sourceforge.net/bowtie2/manual.shtml))
* Run bowtie2 on all files in a directory: **run\_bowtie2.sh**

## Compress and Sort

After alignment has finished, the resultant sam files need to sorted and compressed using:

samtools view -bS <sam\_file> | samtools sort - <bam\_file\_prefix>

The dash (-) tells samtools to take stdin as input.

For instance:

samtools view -bS 020311-1\_S40\_L001\_R1\_001.sam | samtools sort - 020311-1\_S40\_L001\_R1\_001.sorted

Ignore the following warning, it is a known bug.

[bam\_header\_read] EOF marker is absent. The input is probably truncated.

To compress and sort all files in a directory, use **compress\_and\_sort.sh.**

## Call Genotype Likelihoods using Samtools

In this pipeline we will use the Mpileup command in Samtools to call genotype likelihoods or to call variants.

Type the following to call genotype likelihood:

samtools mpileup -g -f <genome> <sorted\_bam\_file> > <bcf\_file>

* -g Compute genotype likelihoods and output them in the binary call format (BCF).
* -f indicate <genome> is indexed by samtools faidx (a module in samtools).

For example:

hg19=/srv/gs1/projects/montgomery/shared/genome/hg19/hg19.fa

samtools mpileup -g -f $hg19 020311-1\_S40\_L001\_R1\_001.sorted.bam > 020311-1\_S40\_L001\_R1\_001.bcf

To inspect bcf files, use bcftools, which comes with samtools.

bcftools view 020311-1\_S40\_L001\_R1\_001.bcf

Additional information:

* bcf file format specification: [link](http://www.1000genomes.org/wiki/analysis/variant-call-format/bcf-binary-vcf-version-2)
* sam file format specification: [pdf](http://samtools.github.io/hts-specs/SAMv1.pdf)
* call genotype over all files in a directory: **call\_genotype\_likelihood.sh**

## Call Variants With Samtools

### One diploid individual:

samtools mpileup -uf ref.fa aln1.bam

bcftools view -bvcg <bcf\_file> > <variants\_bcf\_file>

bcftools view <variants\_bcf\_file> | vcfutils.pl varFilter –D 100 > <filtered\_ variants\_bcf\_file>

samtools mpileup flags

* -u compute genotype likelihood and output in [u]ncompressed format (vcf).
* -f samtools [f]aidx indexed file

bcftools flags

* -b output in bcf format
* -v output variant sites only
* -c call variants using Bayesian inference
* -g call per-sample genotype at variant site

vcfutils flags

* -D 100 controls the maximum read depth, which should be adjusted to about twice the average read depth

### Multiple diploid individuals

TBD

### More information:

* Powerpoint from EBI explaining this pipeline: [pdf](https://www.ebi.ac.uk/training/sites/ebi.ac.uk.training/files/materials/2014/140217_AgriOmics/dan_bolser_snp_calling.pdf)
* batch implementation: **call\_variants\_samtools.sh**

# Call Variants using GATK

## generating index and dictionary

bwa index –a bwtsw reference.fa # generates several files

samtools faidx reference.fa # generates reference.fa.fai

java -jar CreateSequenceDictionary.jar \

REFERENCE=reference.fa \

OUTPUT=reference.dict # generates reference.fa.dict

## Preprocessing

### Trimming

Trim adapters and low quality bases with trim galore!: **run\_trim\_galore.sh**

### Alignment

Align with BWA using:

bwa mem -M -R ’<read group info>’ reference.fa raw\_reads.fq > aligned\_reads.sam

The ‘<read group info>’ can be:

@RG\tID:group1\tSM:sample1\tPL:illumina\tLB:lib1\tPU:unit1

Batch implementation: **run\_bwa\_single\_end.sh**

### Dedupping

Use Picard. This step is only necessary for whole-genome sequencing; you don’t need to dedup for mmPCR-seq data.

### Indel Realignment

TBD

### Base Recalibration

TBD

### Compress And Sort

Use **compress\_and\_sort.sh**

### Index Bam Files

## Variant Discovery

TBD

## Troubleshoot

* SAM file doesn't have any read groups defined in the header. The GATK no longer supports SAM files without read groups

Many GATK algorithms needs information that certain reads are sequenced together as these algorithms try to compensate the variation from one sequencing to the next. A read group, or RG, provides this information. Reads in the same read group are assumed to be sequenced on one lane. ([ref](https://www.biostars.org/p/43897/))

Implementation: **add\_read\_groups.sh** to add read group.

Click [here](https://www.broadinstitute.org/gatk/guide/article?id=2909) to know more about how to fix a badly formatted BAM.

TODO: add batch functionality to **add\_read\_groups.sh**

* Input files reads and reference have incompatible contigs: Relative ordering of overlapping contigs differs, which is unsafe.

Solution: reorder the sam/bam file with picard/ReorderSam. ([ref](https://www.broadinstitute.org/gatk/guide/article?id=58), [help](http://broadinstitute.github.io/picard/command-line-overview.html#ReorderSam))

Implementation: **reorder\_sam.sh**

TODO: add batch functionality to reoder\_sam.sh

# FastQ Quality Control:

First, let us check the quality of the sequencing result using fastQC.

Let’s say your file is located in <input\_directory>. To make our life easier, let’s create an atlas.

*in=<input\_directory>*

*out=<output\_directory>*

To run fastQC, first let us load the module (I’m on a cluster, but you can also download fastQC from [here](http://www.bioinformatics.babraham.ac.uk/projects/fastqc/).

*module load fastqc/0.11.2*

To run, type:

*fastqc $in/<input\_file> --outdir=$out*

Now we will have our results in the output directory. The document from [here](http://www.bioinformatics.babraham.ac.uk/projects/fastqc/Help/3%20Analysis%20Modules/) will help you understand the result.

* Batch implementation: **run\_fastqc.sh**

## Adapters Trimming

If you see adaptor contamination especially towards the end. This is because if you template is not long enough, the adapter sequences will be sequences right after the template ends. If this is the case, we will need to cut the adatpters with cutadapt.

If you don’t have cutadapt, install using:

*pip install cutadapt*

To trim illumina single-ended indexed libraries:

*cutadapt -a GATCGGAAGAGCACACGTCTGAACTCCAGTCACNNNNNNATCTCGTATGCCGTCTTCTGCTTG -o trimmed.1.fastq reads.1.fastq > -o trimmed.1.log*

# note to self: Rui’s mmPCR adapters has an additional ‘A’ at the beginning. So the read will be *AGATCGGAAGAGCACACGTCTGAACTCCAGTCACNNNNNNATCTCGTATGCCGTCTTCTGCTTG*

To run cutadapt on all files in a folder, use the script **run\_cutadapt.sh.**

If you don’t understand why adapters need to be trimmed, look [here](http://www.med.unc.edu/pharm/calabreselab/files/tufts-sequencing-primer).

## Adapter and Low Quality Base Trimming

I recommend Trim Galore!, which uses cutadapt to cut adapters and fastQC to automatically produce quality control plots. It also trim low quality bases. You can find its manual [here](http://www.bioinformatics.babraham.ac.uk/projects/trim_galore/trim_galore_User_Guide_v0.3.7.pdf).

A few tricks:

* Use the first 13 bases, give or take, of the adapter sequence instead of the whole Illumina adapter. This tweak not only removes whole adapter but also adapter fragments.
* Set the minimum quality score to 30 instead of 20. This results a little bit more loss but the quality increase is tremendous.

# mmPCR data processing

## Overview

1. fastQC

2. trim adapters and low quality bases

3. mapping

4. compress and sort

5. pileup

6. parse\_pileup: Count Read Depth for Variant Site

7. extract\_from\_zpileup: select sites of interests

8. R

## Parse\_pileup: Count Read Depth for Variant Site

First pileup (refer to "Samtools::Converting Bam Format to Pileup Format").

Then run parse\_pileup.py

cat <bam> | python parse\_pileup.py > <zpileup>

* Run over a directory: **run\_parse\_pileup.sh**

# Testing

## Testing on a Subset of Data

Testing your scripts on a small dataset can save you a lot of time. You don’t want to wait days to see a job fail. If there is a bug in your code, you want it to fail quickly.

There is no standard procedure in testing your script, but here are the procedures I use:

1. mkdir $test # you want to run the test script in a separate directory.
2. head –n 100 <dataset> > $tests/<dataset.test> # take a small subset of your data
3. run your script on <dataset.test>.

Simple, right?

# Documenting Code and Equations with Microsoft Word

Although many tools designed for documenting code and equations are out there, word is still a good way to communicate in a fast and efficient way.

## Remove the annoying squiggly underlines:

* Go to 'spelling and grammar…'
* Click 'options'
* Check 'Hide grammar errors in this document' and 'Hide spelling errors in this document'.

## Disable smart quote:

* Go to 'spelling and grammar…'
* Click 'options'
* Click on 'show all'; go to 'autocorrect'
* Uncheck 'straight quotation marks' to 'smart quotation marks'.

# Reference

# Bed Specification

## SNP representation

chrom SNP\_pos -1 SNP\_pos

# Fastq Specification

## 'B' at the end of reads

From Illumina 1.5+, reads with < Q15 is replaced by 'B' (Q2), indicating these reads should not be used in downstream analysis.

# hg19

## Lower Case vs Upper Case in HG19

Repeats from RepeatMasker and Tandem Repeats Finder (with period of 12 or less) are shown in lower case; non-repeating sequence is shown in upper case.

# Sam Specification

* @PG: records of programs that processed the sam file.
* Full specification: [link](http://samtools.github.io/hts-specs/SAMv1.pdf)