Bioinformatics Protocols

Viriya Keo

2024-06-20

Table of contents

Preface

This is a summary of commonly used pipelines and data analysis protocols used in the Yu Lab. Furthermore, it includes many the tips and tricks I've learned and kept track of. I consider this a mini-cookbook of sorts.

This book is not an exhaustive list of everything we do in the lab. For example, I do not have any experience with DiMeLo-Seq data analysis. I also do not consider this the best practices or the only possible way of doing things, but I've tried my best to pick the best/easiest methods. However, the code for book is published on GitHub and any lab member can contribute to it in the future and add more sections or change approaches.

I've tried to follow the typical life cycle of NGS data analysis in organizing the book. We start with basic QC and processing pipelines. The chapters following are method specific analysis. At the end, I have programming tips as well as general resources that have helped me immensely in learning everything in this book. More specific resources are provided where relevant.

I try to keep this short and not include all of the background for certain decisions made. Some of the code included may seem very trivial but I would've wished I had the handy snippets when I started out. I also assume you already know some basics.

This book was written in Quarto. To learn more about Quarto books visit https://quarto.org/docs/books.

1 Glueing Together Pipelines

This chapter is named as such because I will be detailing the processing pipelines for our NGS data and the command line, i.e. the glue, that connects different parts together.

As of October 2023, we are starting to move ahead with switching to Nextflow and nf-core's processing pipelines which will take care of most basic processing steps such as mapping so I will not list them in too much detail.

1.1 Working with High Performance Computing (HPCs)

Northwestern's Quest User Guide is a great place to start with learning how HPCs work. However, with our own server at Emory, things are much simpler.

Cancel a lot of pending jobs `scancel --state PENDING --user vkp2256`.

1.2 Environment Reproducibility

Many packages will now use conda, so many of those will be self-contained. I highly recommend using mamba to speed up installations.

For R, while one can use a conda environment, I've found it to be hard to set it up, so try to use only one version of R in one project to avoid dependency issues and do `sessioninfo()` to get package versions that were installed.

1.3 Nextflow

Nextflow is a pipeline construction language. Nf-core is a publicly contributed best-practices of bioinformatics pipelines. If you want to do basic, routine analysis, using nf-core's ready-made pipelines is best, but because of its large code-base and complexity, it might be hard to debug. You may want to start your own nextflow container if you find yourself doing similar analyses many times.

Use `nextflow pull nf-core/rnaseq` to update pipelines. When running the pipeline after this, it will always use the cached version if available - even if the pipeline has been updated since.

1.4 The Command Line

It is essential to master the basics of the command line. (Buffalo 2015, chap. 8) is a great place to get started. I found learning about path expansion, awk, and vim extremely useful. The Missing Semester is also a great resource.

Make your executable bash script have a help page by creating a Help facility.

While I think going through these well-documented resources is better than anything I could share, I'll document the things I found really useful here.

> ⚠ Warning
>
> Some of these might not be accurate, test carefully before using them (as you should do with any random piece of code you find).

1.4.1 One-liners

There are many great bioinformatics one-liner compilations such as this one. I suggest you keep a list of your own that you find yourself re-using often.

1.4.2 File manipulation

Read in files line by line and perform an action with it.

```
file = "/path/to/yourfile.txt"
while IFS= read -r line
do
echo "$line"
echo "${line:8:5}" # grab chars starting at 8th position(0 index) for 5 chars
done <"$file"
```

If you want to refer to columns in your file

```
# Loop through each line in the mapping file and rename the files
while IFS=' ' read -r old_name new_name; do
```

```
    # Rename the files
    mv "$old_name" "$new_name"
done < "$mapping_file"
```

Checking if a directory already exists

```
if [ ! -d $dir ]; then
    mkdir $dir
fi
```

For loops: there are many ways of specifying what to loop through

```
#!/bin/bash
for i in m{1..3} m{13..15}
for i in 1 2 3 4 5
for i in file1 file2

do
echo "Welcome $i times"
j="$(basename "${i}")" # get the file name

k="$(basename "${i}" | sed 's/_[^_]*$//')" # prints file name without the extension and th
l="$(dirname "${i}")" # prints the dirname without filename
echo raw/JYu-${i}_*_L001* # filename expansion

done
```

For more complex naming schemes, you may use an array.

```
# Specify the last two digits of the numbers
numbers=("59" "82" "23" "45" "77")

# Iterate over the array
for num in "${numbers[@]}"; do
  echo "SRR13105$num"
done

# Iterate over indexes
for ((index=0;index<5;index++))
do
num="SRR13105${numbers[$index]}"
done
```

See this for more examples and use cases.

1.4.3 Unix Text Processing: Awk and Sed

Awk

Awk statements are written as `pattern { action }`. If we omit the pattern, Awk will run the action on all records. If we omit the action but specify a pattern, Awk will print all records that match the pattern.

```
awk '{if ($3 == "sth) print $0;}' file.tsv
awk -F "\t" '{print NF; exit}' file.tsv # prints number of fields/columns
awk -F "\t" 'BEGIN{OFS="\t";} {print $1,$2,$3,"+"}' input.bed > output.bed # indicate that
color="$(awk 'NR==val{print; exit}' val=$line color_list_3_chars.txt)" # pick line based o
```

Sed

Sed statements are written as `substitute: 's/pattern/replacement/'`

sed -e `some rule` -e `another rule` The g/ means global replace i.e. find all occurrences of foo and replace with bar using sed. If you removed the /g only first occurrence is changed. chaining rules also possible: sed -e 's/,//g ; 5q' print only the first 5 lines

1.4.4 Miscellanous

- Inside a bash script, you can use `$0` for the name of the script, `$1` for the first argument and so on.

- `$?` show error code of last command, 0 if everything went well, `$#` number of commands.
- `shuf -n 4 example_data.csv` print random 4 lines.

- `nohup command &` will make commmand run in the background, `fg` to bring it back to the foreground.

- https://www.shellcheck.net/ is a website where it can check for errors and bugs in your shell scripts.
- `paste -sd,` concatenate lines into a single line -s, delimited by a comma

1.4.4.1 Working with lines

count number of occurrences of unique instances

```
sort | uniq -c
```

output number of duplicated lines if there are any

```
uniq -d mm_gene_names.txt | wc -l
```

count fastq lines (assuming well-formatted fastq files which it usually are)

```
cat file.fq | echo $((`wc -l`/4))
```

print only the number of lines in a file and no filenames

```
wc -l m*/m*.narrowPeak | cut -f 4 -d' '
```

to get rid of x lines at the beginning of a file

```
tail -n +2 file.txt # starts from the 2nd line, i.e. removing the first line
```

to see the first 2 lines and the last 2 lines of a file

```
(head -n 2; tail -n 2) < Mus_musculus.GRCm38.75_chr1.bed
```

1.4.4.2 Working with columns

```
cut -f 2 Mus_musculus.GRCm38.75_chr1.bed # only the second column
cut -f 3-8 Mus_musculus.GRCm38.75_chr1.bed # range of columns
cut -f 3,6,7,8 Mus_musculus.GRCm38.75_chr1.bed # sets of columns, cannot be reordered
cut -d, -f2 some_file.csv # use comma as delimiter

grep -v "^#" some_file.gtf | cut -f 1-8 | column -t | head -n 3 #prettify output
```

remove header lines and count the number of columns

```
grep -v "^#" Mus_musculus.GRCm38.75_chr1.gtf | awk -F "\t" '{print NF; exit}'
```

1.4.4.3 Working with strings

to remove everything after first dot. useful for getting sample name from filename.fastq.gz

```
${i%%.*}
```

to remove everything after last dot. useful for getting sample name from filename.param.fastq

```
${i%.*}
```

to remove everything before a /, including it

```
${i##*/}
```

to make uppercase

```
${var^}
```

1.4.4.4 Directories

to get human readable size of folders under the parent folder

```
du -h --max-depth=1 parent_folder_name
```

s means summary

```
du -sh /full/path/directory
```

print each file in new line

```
ls -1
```

check what file takes the most space (d is one level down)

```
du -d 1 | sort -n
```

1.4.4.5 Alignment files

Removing EBV chromosomes for viewing on UCSC browser.

```
samtools idxstats m${i}_sorted.bam | cut -f 1 | grep -v 'chrEBV' | xargs samtools view -h
```

Remove reads not mapping to chr1-22, X, Y. this does not remove from headers. The first sed expression removes leading whitespace from echo (-n to), the second expression to add “chr” at the beginning.

```
samtools view -o mr392_filtered1.bam mr392_sorted.bam `echo -n {{1..22},X,Y}$'\n' | sed -e
```

or you can just chain together reverse grep to remove any chromosomes you want

```
chr=samtools view -H m1076_rgid_reheader_st.bam | grep chr | cut -f2 | sed 's/SN://g' | gr
```

the awk statement is to remove the unknown chromosomes and random contigs since they will be longer than 6 chars

1.5 Bioinformatics Bits

These are mostly still command line tools, but more bioinformatics related.

1.5.1 Getting files from GEO

Using SRAtools https://bioinformaticsworkbook.org/dataAcquisition/fileTransfer/sra.html#gsc.tab=0 To get the SRR runs numbers, use the SRA Run Selector. Select the ones you want to download or all of them, and download the Accession List. Upload to quest. Better to do a job array for each SRR. fasterq-dump is now the preferred and faster option. To download bam, can bystep sam step.

```
module load sratoolkit
fastq-dump --gzip SRR # use --split-files for paired-end
fasterq-dump SRR -O output/dir # for 10x, need to use --split-filles and --include-technic
sam-dump SRR | samtools view -bS -> SRR.bam
```

Using ftp
This can be faster than SRAtoolkit but only works if those files have been uploaded to EBI.

ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR101/006/SRR1016916

pattern: fastq/first 6 chars/00(last number)/full accession

1.5.2 Making UCSC tracks

```
mdoule load homer
makeTagDirectory tag1045 1045.bed
makeUCSCfile tagm57 -o m57 -norm 2e7v
```

With RNA-Seq UCSC tracks, use –fragLength given, otherwise it messes up the auto-correlation thinking that's it's ChIP-Seq leading to screwed up tracks.

You can upload the tracks and save it as a session that can be shared.

1.5.3 samtools

Learn the Bam Format.

```
samtools view -h test.bam # print bam with headers
samtools view -H test.bam # headers only
samtools flagstat test.bam # get mapping info
samtools idxstats test.bam
samtools -f 0x2 # read mapped in proper pair
samtools view -H test.bam | grep @HD # Check if BAM files are sorted
samtools view -F 4 test.bam | wc -l # check how many mapped reads
```

1.6 R Quirks

When saving anything produced with ggplot and you will be editing in Illustrator, set `useDingbats = FALSE`.

2 Bulk Analysis

This chapter is for the downstream analysis of the most common bulk NGS experiments we perform. For functional analysis that are common to all these methods, see next chapter.

2.1 RNA-Seq

While RPKM has some issues and is not the most correct quantification, for basic comparison, this is good enough to share with biologists. The nf-core pipeline will provide this value by default.

2.1.1 Differential Gene Expression

One of the most basic and common analysis on RNA-Seq. We use DESeq2 and their well documented vignette is worth reading from start to end for the beginner.

> **Note**
>
> While our pipeline originally used the `STAR --quantMode` to quantify genes, with switching to nf-core, we are also switching to STAR alignment followed by RSEM quantification.

```
library(tidyverse)

files <- list.files(path = "./working_dir", pattern = "*genes.results$", full.names = TRUE
rsem_results <- lapply(files, read_delim)
expected_counts_list <- lapply(rsem_results, function(x) { x$expected_count })
expected_counts <- do.call(cbind, expected_counts_list) %>% as.data.frame()
```

RSEM produces non-integer counts, and we can by-pass that by using `round()`. Alternatively, you can use tximport to read the files in.

```
expected_counts <- round(expected_counts)
rownames(expected_counts) <- rsem_results[[1]]$gene_id
colnames(expected_counts) <- stringr::str_extract(files, "mr\\d+") # our RNA-seq samples u
```

```
sampleTable <- data.frame(condition = factor(rep(c("control", "knockdown"), each = 3)),
                          replicate = factor(rep(seq(1,3))))
rownames(sampleTable) <- colnames(expected_counts)
```

DESeq2

```
dds <- DESeqDataSetFromMatrix(expected_counts, sampleTable, design = ~condition)
keep <- rowSums(counts(dds)) > 10
dds <- dds[keep,]
dds <- DESeq(dds)
res <- results(dds, alpha = 0.01)
summary(res)
```

Difference between rlog, vst and lfcShrink https://support.bioconductor.org/p/104615/.

Plotting PCs

DESeq2's `plotPCA()` function will plot the top 500 most variable genes. The chunk below will plot all genes.

```
degenes <- res %>% subset(padj < 0.01)
dds_rlog <- rlog(dds)
pca_data <- t(assay(dds_rlog)) %>% prcomp()
autoplot(pca_data, data = sampleTable, colour = "condition") +
  geom_text_repel(label = rownames(sampleTable))
rlog_de <- assay(dds_rlog) %>% subset(rownames(dds_rlog) %in% rownames(degenes))
rlog_de_scaled <- t(scale(t(rlog_de)))
```

After getting DEGs, you'd want to group the genes into biological functions. See Section 3.2 for Over representation analysis (ORA) with GO and KEGG terms as well as Gene Set Enrichment Analysis (GSEA) with ranked genes.

2.1.2 Deciding Groups and Plotting

While you can use k-means manually to get seperate groups, ComplexHeatmap allows you to do so with more flexiblity and get visualizations as well.

```
mat_colors <- list(
  replicate = c(brewer.pal(3, "Accent")),
  condition = c(brewer.pal(6, "Set1")))
names(mat_colors$replicate) <- unique(sampleTable$replicate)
```

```
names(mat_colors$condition) <- sampleTable$condition

col_anno <- HeatmapAnnotation(df = sampleTable,
                              which = 'col',
                              col = mat_colors
)

hmap <- Heatmap(rlog_de_scaled,
                    name = "scaled",

                    # Row Params
                    show_row_names = FALSE,
                    row_title_rot=0,
                    cluster_row_slices = FALSE,
                    border = TRUE,
                    row_km = 2, # split rows into 2 groups

                    # Column Params
                    cluster_columns = FALSE,
                    column_title = "Rlog Transformed Expression for all DE genes",
                    top_annotation = col_anno)
hmap <- draw(hmap) # assigning so that k-means is only called once
row_order(hmap) # grab the different groups in rows
```

See here for why we assign `draw()`.

2.2 ChIP-Seq and ATAC-Seq

2.2.1 Spike-in normalization

Guidelines on spike-in normalization from ActiveMotif.

1. Perform ChIP combining the Spike-in Chromatin, Spike-in Antibody, test chromatin and test antibody into the same tube for immunoprecipitation. We suggest using the guidelines provided for chromatin and antibody quantities based on the antibody target.
2. Follow ChIP with Next-Generation Sequencing.
3. Map ChIP-seq data to the test reference genome (e.g. human, mouse or other).
4. Map ChIP-seq data to the Drosophila reference genome.
5. Count uniquely aligning Drosophila sequence tags and identify the sample containing the least number of tags.

6. Compare Drosophila tag counts from other samples to the sample containing the least tags and generate a normalization factor for each comparison. (Sample 1 with lowest tag count / Sample 2) = Normalization factor
7. Downsample the tag counts of data sets proportional to the normalization factor determined

2.2.2 Peak calling

Depending on if you use the nfcore's pipeline or your own, you will have to call peaks. I use MACS2 and here are some details I've gathered.
https://github.com/crazyhottommy/ChIP-seq-analysis/blob/master/part1.3_MACS2_peak_calling_details.md

Why I skip model building https://github.com/macs3-project/MACS/issues/391

```
macs2 callpeak -t mxx_sorted.bam --outdir macs/mxx -n mxx -g hs -q 0.01 --nomodel --shift
```

2.2.3 Overlapping peaks

After MACS2 peak-calling, we may want to see how many peaks overlap in different conditions. Even though it's named mergePeaks, you will be able to get overlapping statistics from this. Be mindful of long path names as mergePeaks will produce errors.

```
mergePeaks -d 100 pu1.peaks cebp.peaks -prefix mmm -venn venn.txt
```

> Tip
>
> The -d flag changes the unique peaks to 100 bp each and keep the shared peaks same size.
> To get literal 1bp overlap, just omit -d argument altogether

The easiest way to plot this result is with Vennerable.

```
a <- tot[1]
b <- tot[2]
ab <- tot[3]

venn_obj <- createVennObj(nSets = 2, sNames = c("m1043", "m1044"), # names in order of fir
                      sSizes = c(0, a, b ,ab))
vp <- plotVenn(nVennObj = venn_obj)
```

This gets tedious with more than 2 sets. See code below. You should not make Venn Diagrams with more than 4 sets. An upset plot is better in that scenario.

```
# 3 sets -----
a <- tot[1]
b <- tot[2]
ab <- tot[3]
c <- tot[4]
ac <- tot[5]
bc <- tot[6]
abc <- tot[7]

# 4 sets -------
a <- tot[1]
b <- tot[2]
ab <- tot[3]Ve
c <- tot[4]
ac <- tot[5]
bc <- tot[6]
abc <- tot[7]
d <- tot[8]
ad <- tot[9]
bd <- tot[10]
abd <- tot[11]
cd <- tot[12]
acd <- tot[13]
bcd <- tot[14]
abcd <- tot[15]
```

This is a more automated way of reading in the data, here using 4 sets as an example.

```
venn <- read.table("results/output_nepc.bedpe", header = TRUE, sep = "\t")
venn$alpha <- apply(venn, 1, function(x) {
  sets <- c("A", "B", "C", "D")
  selected_sets <- sets[which(x == "X")]
  paste(selected_sets, collapse = "")
})

order_vector <- c("0", "A", "B", "AB", "C", "AC", "BC", "ABC", "D", "AD", "BD", "ABD", "CD

venn$alpha <- factor(venn$alpha, levels = order_vector)
```

```
# Sort the data frame based on the factor levels
venn <- venn[order(venn$alpha), ]
venn_plot <- Venn(SetNames = c("93", "145.1", "145.2", "nci"),
                  Weight = c(0, venn$Features))
plot(venn_plot, type = "ellipses")
```

Creating upset plots https://github.com/hms-dbmi/UpSetR

```
library(UpSetR)
venn$Sets <- apply(venn[, -1], 1, function(x) {
  sets <- colnames(venn)[2:17]
  selected_sets <- sets[which(x == "X")]
  paste(selected_sets, collapse = "&")
})

upset_input <- c(venn$Features)
names(upset_input) <- venn$Sets
upset(fromExpression(upset_input), nsets = 16, nintersects = 100, number.angles = 45)
```

A sample script to automate this process. I haven't incorporated the above part into this script.

Listing 2.1 `mergePeaksandplot.sh`

```
module load homer/4.10
module load R/4.0.3
mergePeaks control_peaks.narrowPeak purturbed_peaks.narrowPeak -prefix ov -venn venn.txt
# n_way script, venn.txt, last sample label, middle samples label, first sample label, txt
Rscript --vanilla venn2way.R venn.txt purturbed control venn
```

The accompanying R file:

To make Venn Diagrams that are more accurately weighted, use nVennR. Sadly, it looks like it's been removed from CRAN since the last time I've installed it.

If you have replicates, overlapping peaks can be obtained via packages like MAnorm2. Read more about dealing with replicates here.

2.2.4 Motif Enrichment

Homer `mergePeaks` is able to take the narrowPeaks format as input, but if you're doing homer `mergePeaks` first and then `findMotifs`, you need to change the Homer `mergePeaks` output to

Listing 2.2 `venn2way.R`

```
library(Vennerable)

args <- commandArgs(trailingOnly = TRUE)

venn <- read.table(args[1], header = TRUE, sep = "\t")
tot <- venn[[Total]]

a <- tot[1]
b <- tot[2]

ab <- tot[3]

venner <- Venn(SetNames = c(args[3], args[2]), # opposite labelling to nVennR, going from
               Weight = c(0, b, a, ab))
png(paste0(args[4], ".png"))
plot(venner)
dev.off()
```

exclude the header line first. This also applies to deepTools heatmaps.

```
findMotifsGenome.pl peak_or_bed /projects/p20023/Viriya/software/Homer4.10/data/genomes/hg
```

-size 200 is the default, and is calculated from the center of the peaks. If your peaks are bigger and you wish to use the entire region, use `-size given`. However, when the regions are too large, motifs will not be significantly enriched.

2.2.5 Plotting Heatmaps

Here I'm showing a sample deepTools script to plot heatmaps. First you need bigwig files. `bamCoverage` offers normalization by scaling factor, Reads Per Kilobase per Million mapped reads (RPKM), counts per million (CPM), bins per million mapped reads (BPM) and 1x depth (reads per genome coverage, RPGC).

```
bamCoverage --bam m${i}_sorted.bam --outFileName m${i}.bw --normalizeUsing CPM \
--extendReads 200 --numberOfProcessors 6 --binSize 20
```

I find having paths defined at the top of the script makes it less likely to have mistakes.

```
day3="m1015_1055_hg38/foxa2/CPM"
nci="m1015_1055_hg38/CPM"
chip="../Viriya/analysis/foxa2/chip-seq"

computeMatrix reference-point --referencePoint center -S \
$day3/m674.bw $day3/m676.bw $tf/m914.bw $tf/m921.bw $nci/m1043.bw $nci/m1044.bw \
-R $chip/clustering/output/1.bed \
$chip/clustering/output/2.bed \
$chip/clustering/output/3.bed \
$chip/clustering/output/4.bed \
-a 3000 -b 3000 -o $chip/heatmaps/scaled_FOXA2_sites.npz \
--samplesLabel m674 m676 m914 m921 m1043 m1044 \
-p max --blackListFileName $chip/ENCFF356LFX.bed.gz

plotHeatmap -m $chip/FOXA2_6_samples/heatmaps/scaled_FOXA2_sites.npz \
-o $chip/FOXA2_6_samples/heatmaps/scaled_FOXA2_sites.pdf \
--colorMap Blues
```

`--sortUsingSamples` is not 0 indexed. 1st sample is 1. `--sortUsingSamples` is also usable in plotHeatmap so we can save that to there for more flexible plotting.

`--regionsLabel`: "xxx binding sites n=2390" put it in quotes if you don't want to key in escape too many times

`--sampleLabels`: if you're sure you're not going to change any labels, put it in makeheatmap step because fewer things to type when redoing the heatmap multiple times due to scaling etc, therefore fewer mistakes. otherwise put it in plotHeatmap.

In legends: "Pol II" spaces and parantheses need to be escaped, but not + signs.

Use BED6 format annd not the BED3 if you care about strandedness https://github.com/deeptools/deepTools/issues/886.

To get sorted output bed files, use the `--sortedOutRegions` at the plotHeatmap step, not computeMatrix.
Use `--clusterUsingSamples` for a more robust delineation.

2.2.6 Genomic Annotation

ChIPSeeker

```
library(org.Hs.eg.db)
library(TxDb.Hsapiens.UCSC.hg38.knownGene)
```

```
promoter <- getPromoters(TxDb=txdb, upstream=3000, downstream=3000)

mxxx <- readPeakFile("/path_to_narrowPeak/mxxx_peaks.narrowPeak")
tagMatrix <- getTagMatrix(mxxx, windows = promoter) # usually needs an interactive job
peakAnno <- annotatePeak(mxxx, tssRegion = c(-3000,3000), TxDb = txdb, annoDb = "org.Hs.eg
plotAnnoPie(peakAnno, main = "mxxx", line = -6)
vennpie(peakAnno, r = 0.1)
upsetplot(peakAnno) + ggtitle("mxxx")
plotAnnoBar(peakAnno)
```

`annotatePeak` will default to chosing one gene per region. This is fine for small binding sites but for larger regions use `seq2gene` as it will map genomic regions in a many-to-many manner.

3 Functional Analysis

What do do once you have a list of genes?

3.1 Gene name conversions

Getting genomic coordinates from a list of gene names.

```
ensembl <- useEnsembl(biomart = "ensembl", dataset="hsapiens_gene_ensembl", version = 86)
chr <- c(seq(1,22), "X", "Y")

gene_coor <- getBM(attributes = c("hgnc_symbol", "chromosome_name", "start_position", "end
                   filters = "hgnc_symbol", values = rownames(res_sh_vs_ctrl), mart = ens
                   useCache = FALSE)
# patched chromosomes are coming up, therefore not unique, need to filter out
gene_coor <- gene_coor[gene_coor$chromosome_name %in% chr,]
```

There are many ways of doing this, they're all slightly different depending on the database.

3.2 Enrichment Analysis

Many of the functional analysis is achieved using ClusterProfiler. While the vignette is very comprehensive, it can be overwhelming to see which functions are useful, so I will point out the most useful ones here.

3.2.1 Overrepresention Analysis (ORA)

A very good explanation and accompanying paper of why we need background lists. Which genes to use for backgrounds? Check this.

```
genes <- as.data.frame(peakAnno)$geneId # get the genes from peak annotation

geneUniverse <- rownames(res_lrt) # all genes that were tested for significance
```

```
go <- enrichGO (gene = genes,
          keyType = "SYMBOL",
          universe = geneUniverse,
          ont = "BP",
          OrgDb = org.Hs.eg.db,
          qvalueCutoff = 0.05)


go_simplified <- simplify(go) # to get rid of redundant GO terms
```

To wrap long labels into a few lines

```
dotplot(go) + scale_y_discrete(labels = function(x) str_wrap(x, width = 30)) + ggtitle("")
  theme(plot.title = element_text(hjust = 0.5))
```

To use KEGG, you need to convert gene symbols to entrezid or use the geneId column from annotatePeaks.

```
deg_list_entrez <- lapply(deg_list, function(x) {
  mapIds(org.Hs.eg.db, rownames(x), "ENTREZID", "SYMBOL")
})
kegg <- enrichKEGG(gene = deg_list_entrez$up, organism = "hsa")
```

Or use bit_kegg https://guangchuangyu.github.io/2016/05/convert-biological-id-with-kegg-api-using-clusterprofiler/. I have not tried this method yet.

3.2.2 Gene Set Enrichment Analysis (GSEA)

Explanation of GSEA and how to do it in R https://sbc.shef.ac.uk/workshops/2019-01-14-rna-seq-r/rna-seq-gene-set-testing.nb.html.
How I first did it https://stephenturner.github.io/deseq-to-fgsea/.

I rank the genes by the stat column. For the Wald test, stat is the Wald statistic: the log2FoldChange divided by lfcSE.

```
ranked_genes <- degenes %>% as.data.frame() %>% rownames_to_column("symbol") %>% dplyr::se
ranked_genes <- ranked_genes %>% arrange(-stat) %>% deframe()
```

We usually perform GSEA with the Molecular Signature Database and there is a package for easy retrieval.

```r
library(msigdbr)
h <- msigdbr(species = "Homo sapiens", category = "H") %>%
  dplyr::select(gs_name, gene_symbol)
gsea <- GSEA(ranked_genes, TERM2GENE = h)
ridgeplot(gsea)
enrichplot::gseaplot2(gsea, geneSetID = "HALLMARK_")
```

4 3D Chromatin Organization

Hi-C is a method to capture chromatin contacts genome-wide. Read through the resources for more background. Micro-C is very similar except it does not use a restriction enzyme, leading to better capture. Most analysis are the same for both and I will point out where they differ.

4.1 Resources

Comparative study on chomatin loop callers

4.2 Processing

We have processed all our samples with runHiC from Feng Yue's lab as that was my committee member and expert on HiC. Many people use HiC-Pro but I have tested it and it is much slower. The one time I tried nf-core's HiC pipeline for Micro-C, it failed.

Each sample will require it's own folder, with this structure.

sample
    data
        hg38
        HiC-gzip
    workspace
        datasets.tsv

All fastq files will be in HiC-gzip, with _R1.fastq.gz and _R2.fastq.gz changed to _1.fastq.gz and _2.fastq.gz. I haven't found a workaround to avoid the renaming. The datasets.tsv, technical replicates and biological doesn't really matter but if you want the end file to combine everything, add rep1, rep2 per pair of fastq files so the final resulting file will be denoted allReps. Do `runHiC quality` to get a sense of how well the experiment worked after shallow sequencing.

4.3 Downstream analysis

Many of these have many tools available to call them. I have attempted to choose the most well-supported tools in the same ecosystem to avoid unnecessary format changes and hidden mistakes. I ended up with the Open2C ecosystem's cooltools and many of Dr. Feng Yue's lab's tools.

4.3.1 A/B Compartments

I'm following the cooltools compartments tutorial.
Compartments are not comparable between cell types if done using PCA. You can use cscore if a more robust definition is needed. However, the PCA definition is currently still widely used. For this, you need too calculate expected values first and you should output those as a tsv because you'll need it for other analysis.

```
lucap_35cr_cis_eigs = cooltools.eigs_cis(
                        lucap_35cr,
                        gc_cov,
                        n_eigs=3,
                        )
lucap_35cr_eigenvector_track = lucap_35cr_cis_eigs[1][['chrom','start','end','E1']]
```

4.3.2 Topological Associating Domain

There are many TAD calling tools and most of them do not have a great degree of overlap. I chose the insulation score methods in cooltools.

Creating TADs from insulation scores https://github.com/open2c/cooltools/issues/453.

```
windows = [3*resolution, 5*resolution, 10*resolution, 25*resolution]
samples = {"LuCaP70CR" : lucap70cr_insulation_table,
           "LuCaP77CR" : lucap77cr_insulation_table,
          "LuCaP1451" : lucap1451_insulation_table}
for window in windows:
    for table_name, table in samples.items():
        print("sample", table_name, "at", window)
        insul = table[["chrom", "start", "end", f"log2_insulation_score_{window}"]]
        insul.to_csv(f"results_{window}/{table_name}_insulation_scores_{window}.bed", head

        tads = bioframe.merge(table[table[f"is_boundary_{window}"] == False])
```

```
        tads = tads[(tads["end"] - tads["start"]) <= 1500000].reset_index(drop=True) # dro
        tads.to_csv(f"results_{window}/{table_name}_TADs_{window}.bed", header=False, inde
```

4.3.3 Loops

4.3.3.1 Mustache

Use mustache for loop calls of individual samples, and diffMustache for pair-wise comparisons. The default d for diffMustache is 2,000,000bp. Call at various cut-offs with `-pt2` and decide on a reasonable number of loops.

4.3.3.2 Working with loops

homer merge2Dbed.pl will combine seperate loop calls (also TADs if specified in parameter) that are at adjacent pixels (or within distance specified) into a big loop. This will also produce condition specific loops that are just based on coordinates, not statistical difference.

Currently, I'm using LoopRig as the underlying data structure for working with bedpe files. It's really just 2 Granges objects linked together and I've re-written many of their functions. It's most likely unnecessary to use this package.

4.3.4 APA

There are also many implementations of APA calculation. I use coolpup.py, an extension of `cooltools pileup`.

4.4 HiGlass

You either start with a local installation of Docker, or use the online browser at resgen. They have some slight differences.
For HiGlass, you need to ingest most files in a two-step manner, whereas with resgen you can directly upload them (with the exception of TADs). It's better to write scripts to facilitate and automate the ingestion.
The files will not line up perfect with the heat maps due to how HiGlass interprets coordinates. See https://github.com/higlass/higlass/issues/1051. Since HiGlass doesn't have a notion of a genomic assembly, you need to ingest a chromosome size file once when you start.

4.4.1 Docker

Install docker following these instructions.
To view HiGlass on docker locally, start the docker app, run `higlass-manage start` on your terminal. Go to http://localhost:8989/app to see the browser. Deletion of ingested tracks needs access to the superuser via http://localhost:8989/admin/. Once data is ingested, it's kept at a different location and you can usually delete the original files in your local computer to save space.

4.4.2 Resgen

You can create separate projects. Files uploaded needs to be manually tagged. Most important are `assembly:hg38`, `datatype`, and `filetype`. To use gene search, both chromosome info and gene annotations need to be added to the panel using the search bar.

4.4.3 Navigating HiGlass

You need to be able to reproduce each view so you won't have to start from scatch each time. In Docker, you save the View Configs files manually. In resgen, you can save a view.

While the GUI is sufficient, some things are faster and easier changed via the view configs. To change 1D scales, https://github.com/higlass/higlass/issues/946.

4.4.4 Ingesting files specifications

Instructions can be found here but it's sparse and somewhat confusing so I'm including common use cases.

After aggregation (whether that's needed depends on the filetype), ingest files like this with appropriate tags.

```
higlass-manage ingest file --filetype bed2ddb --datatype 2d-rectangle-domains --project-na
```

4.4.4.1 mcool

The .mcool files themselves are easily recognized with just `higlass-manage ingest sample.mcool --assembly hg38`.

4.4.4.2 AB compartments

These files need to be converted into bigwigs first. They are more well supported than bedgraph files. Use UCSC's bedGraphtoBigWig.

4.4.4.3 TADs

For resgen, TADs need to be aggregated first, and the aggregated file is uploaded.

```
j=${1%%.*}

clodius aggregate bedpe \
    --assembly hg38 \
    --chr1-col 1 --chr2-col 1 \
    --from1-col 2 --to1-col 3 \
    --from2-col 2 --to2-col 3 \
    --output-file ${j}.beddb \
    $1
```

Specify `track-type:linear-2d-rectangle-domains` and `filetype:bed2ddb` for top section. For viewing in center section `datatype:2d-rectangle-domains`.

4.4.4.4 Loops

Loops view as arcs currently only works on resgen and not the local docker. Loops on the 2D heatmap will work with docker.

4.4.5 ChIP-Seq and ATAC-Seq tracks

These are just bigwig ingestion.

5 Single-Cell Analysis

Most of these code are reliant on Seurat. You may now and then have to use scanpy in python, or revert to the SingleCellExperiment object utilized in scater.

5.1 Useful links and resources

Fine-tuning UMAP visualizations https://jlmelville.github.io/uwot/abparams.html
TSNE animation https://distill.pub/2016/misread-tsne/
Simple explanantion on TSNE https://www.cancer.gov/about-nci/organization/ccg/blog/2020/interview-t-sne
Different integration methods https://swaruplab.bio.uci.edu/tutorial/integration/integration_tutorial.html
Illustration of how UMAPs can be misleading https://pair-code.github.io/understanding-umap/ https://blog.bioturing.com/2022/01/14/umap-vs-t-sne-single-cell-rna-seq-data-visualization/

Single cell best practices from the Theis lab https://www.sc-best-practices.org/preamble.html https://broadinstitute.github.io/2019_scWorkshop/index.html

Deep explanation of Seurat's AddModuleScore function https://www.waltermuskovic.com/2021/04/15/seurat-s-addmodulescore-function/

5.1.1 Psudotime analysis

https://broadinstitute.github.io/2019_scWorkshop/functional-pseudotime-analysis.html
List of single cell pseudotime packages publishe#d https://github.com/agitter/single-cell-pseudotime

5.1.2 Spatial Transciptomics

https://scimap.xyz/tutorials/md/spatial_biology_scimap/

5.1.3 Downloading data

bam files uploaded to SRA will be missing certain 10x flags and will not work. You need to get the original bam files via SRA Data Access Tab and downloaded via wget and run bamtofastq. for 10x, need to use –split-filles and –include-technical.

5.1.4 CellRanger

5.2 Seurat

5.2.1 Integration

- applying sctransform to each sample in the list
- selecting integration features,
- finding integration anchors,
- integrate data at this point, DefaultAssay is integrated, and there is an associated VariableFeatures, which really is output of SelectIntegrationFetures. rownames(integrated[["SCT"]](**scale.data?**)) is the same as SelectIntegration Features again.
  https://github.com/satijalab/seurat/issues/6185 explanation of the different variable gene options from sct integrated models

After `merge()` variable features that have been calculated by SCTransform is reset.

5.2.2 Plotting

Use scCustomize for improved plotting over Seurat's defaults.

To plot heatmaps with extra metadata information, use the scillius package.

To get statistics on differential gene levels, use the package `ggbetweenstats` from ggstatsplot.

```
cell_expression <- rna_integrated@assays$RNA@data[c("MIPOL1", "ETV1", "DGKB", "FOXA2", "AR
meta_data <- rna_integrated@meta.data
plot_data <- cbind(meta_data, cell_expression)
saveRDS(plot_data, "scRNAseq_violin.RDS")
```

5.2.3 Utility functions

Renaming all identities

```
cell_types <- c("0" = "", "1" = "", "2" = "", "3" = "", "4" = "", "5" = "", "6" = "")
Idents(merged) <- "SCT_snn_res.0.2"
merged <- RenameIdents(merged, cell_types)
merged$status <- Idents(merged)
DimPlot_scCustom(integrated, group.by = "cell_types", colors_use = polychrome_pal)
merged$status <- factor(merged$status,
                        levels = c("")) # useful for rearranging levels
Idents(merged) <- "status"
```

Renaming a few identities

```
new_groups <- case_when(seu_obj$clust == "a" ~ "Tumor",
          .default = seu_obj$clust) %>% as.factor()
names(new_groups) <- names(seu_obj$clust)
seu_obj$new_groups <- new_groups
seu_obj$new_groups <- factor(seu_obj$new_groups,
                        levels = c("")) # have to do this again
```

Adding new clusters, cluster 13 was not present previously

```
seu_obj$manual_clusters <- seu_obj$integrated_snn_res.0.6
levels(seu_obj$manual_clusters) <- c(levels(seu_obj$manual_clusters), "13")
seu_obj$manual_clusters[names(seu_obj$manual_clusters) %in% select_cells] <- "13"
```

6 Books and Resources

Here is a list of more comprehensive resources (i.e. not a one off blog or tutorial). I've tried to include free resources here.

> Tip
>
> The most useful resource you have is Google. Get good at googling errors. Get good at finding vignettes and tutorials. Get good at documenting the solutions and new knowledge that you find.

6.1 Bioinformatics

The single most useful book in my bioinformatics career up to date is Bioinformatics Data Skills Buffalo (2015).

I have not read Computational Genomics with R yet, but it looks extremely useful and actually might be a better summary of Chapter 2

The Babraham Institute has a very comprehensive training materials
Single-Cell Best Practices from the Theis Lab.

Epigenomics Workshop by National Bioinformatics Infrastructure Sweden.

https://bioinformaticsworkbook.org/list.html#gsc.tab=0
https://docs.gdc.cancer.gov/Data/Introduction/

6.2 Programming

6.2.1 R

The best introduction book to R is unambigously https://r4ds.hadley.nz/. "The R Graph Gallery boasts the most extensive compilation of R-generated graphs on the web" https://r-graph-gallery.com/index.html

6.2.2 Python

https://wesmckinney.com/book/ https://www.rebeccabarter.com/blog/2023-09-11-from_r_to_python

6.3 Git

https://happygitwithr.com/ is a comprehensive resource on how to use Git with R.

If you want to understand how git works more intuitively, watch this.

6.4 Statistics

StatQuest is a very accessible resource to those without a math background. I would recommend the Statistics Fundamentals and High Troughput Sequencing playlists.

I haven't read Modern Statistics for Modern Biology yet but it looks to be very relevant.

6.5 Miscellanous

The Missing Semester of Your CS Education shows you how to master the command line and vim.
Adobe Illustrator for Scientific Figures from the Raj Lab. [IGV handbook]https://www.igv.org/workshops/BroadApril2017/IGV_SlideDeck.pdf.

References

Buffalo, V. 2015. *Bioinformatics Data Skills: Reproducible and Robust Research with Open Source Tools*. O'Reilly Media. https://books.google.com/books?id=XxERCgAAQBAJ.