SPARTA-teaching Documentation

Release 1.0

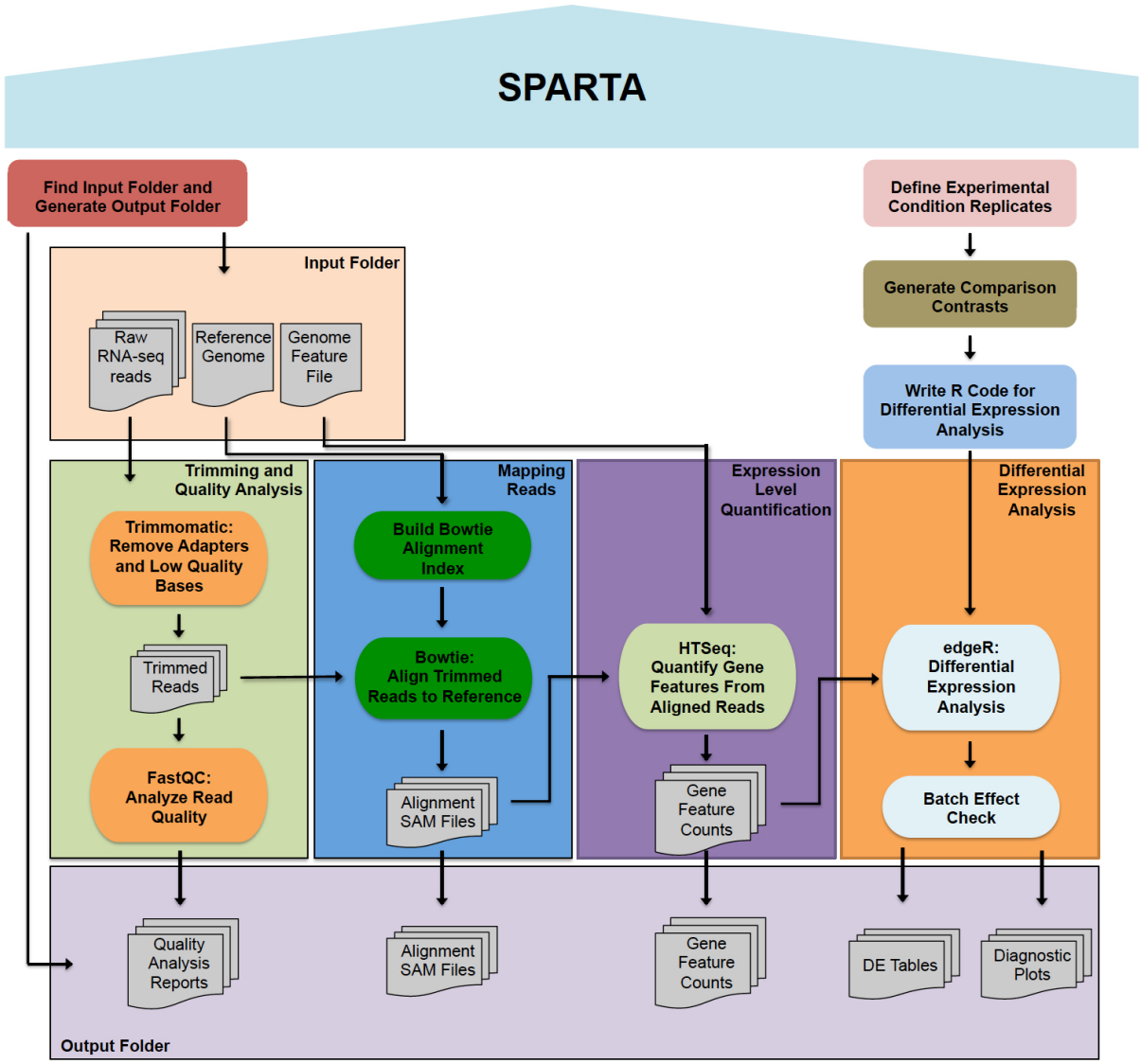
Benjamin K. Johnson

July 02, 2015

Contents

SPARTA is a workflow aimed at analyzing single-end Illumina RNA-seq data. The software is supported on Windows, Mac OS X, and Linux platforms. The workflow combines several tools: Trimmomatic (read trimming/adapter removal), FastQC (read quality analysis), Bowtie (mapping reads to the reference genome), HTSeq (transcript/gene feature abundance counting), and edgeR (differential gene expression analysis). Within the differential gene expression analysis step, batch effects can be detected and the user is warned of the potential, unintended additional variable. The analysis procedure is outlined below.

However, before we can dive into doing the data analysis with our own data or some example data it is worth having a look at some *background information* first.



CHAPTER 1

How to get and use SPARTA:

Mac Users - Mac OS X tutorial

Windows Users - Windows tutorial

Linux Users - Linux tutorial

1.1 Contents:

1.1.1 RNA-seq background information, data analysis procedure, and details of the analysis tools

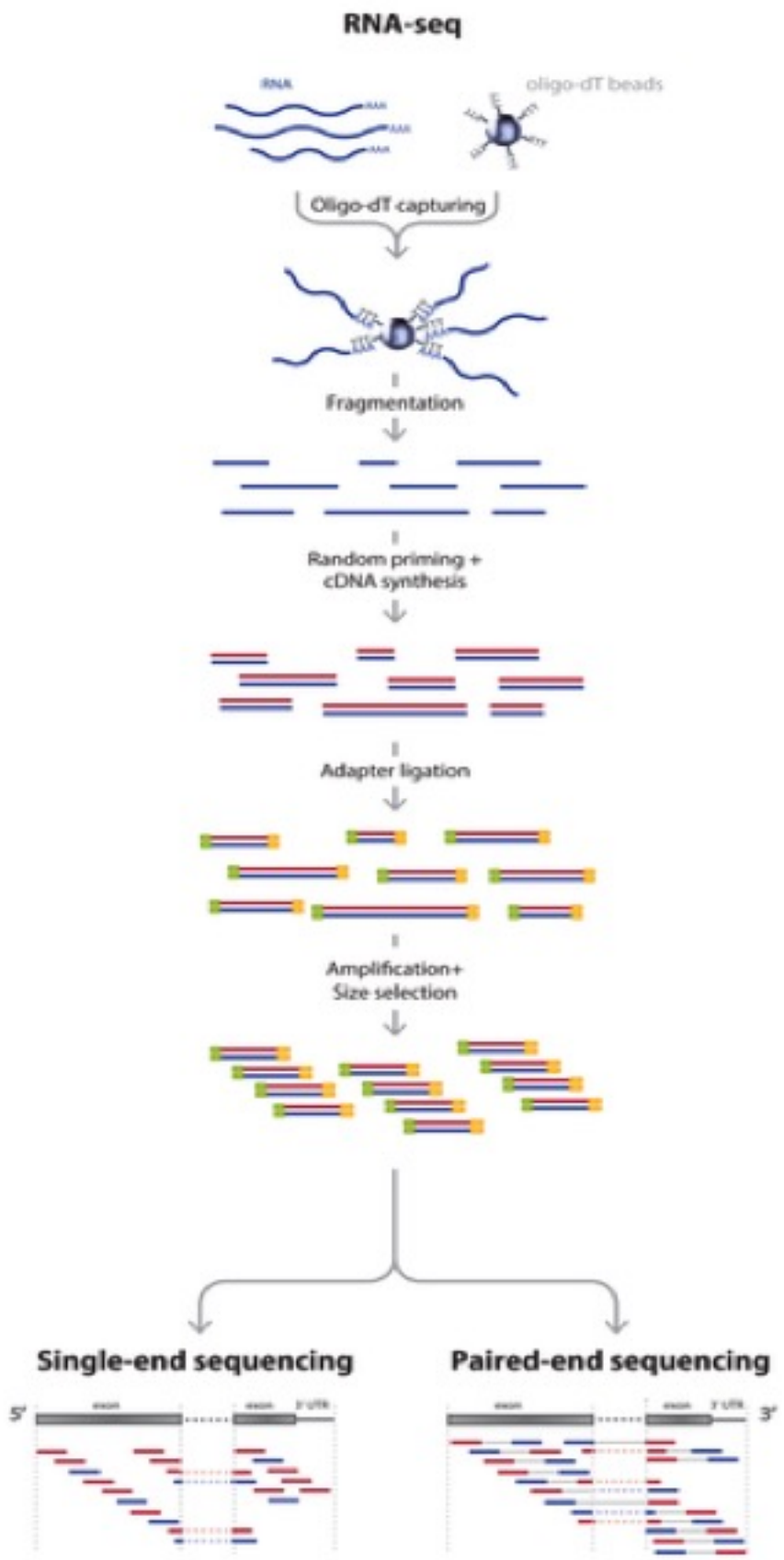
Before we dig into the data and begin trimming and aligning the reads to the genome, I think it is useful to understand what happens *after you submit your RNA to the sequencing facility*. This sort of knowledge can be very useful in understanding what could potentially provide bias and any number of issues to the end dataset. In this session we will cover several things including:

1. *RNA-seq background information*
2. *Basic analysis procedure*
3. *Trimmomatic*
4. *FastQC*
5. *Bowtie*
6. *HTSeq*
7. *Differential gene expression with edgeR*

RNA-seq background information

Before we begin, let's watch a video about how Illumina sequencing works.

This video does a pretty good job explaining how, in generalities the sequencing process works for DNA. So for sequencing RNA, the process is as follows:


RNA-seq
RNA
oligo-dT beads
Oligo-dT capturing
Fragmentation
Random priming +
cDNA synthesis
Adapter ligation
Amplification+
Size selection
Single-end sequencing
Paired-end sequencing
5'
3'

Adapted from: Zhernakova et al., PLoS Genetics 2013

So actually, we aren't sequencing RNA at all! We are sequencing the cDNA made from the RNA. RNA-seq is a high resolution next generation sequencing (NGS) method to assess the transcriptome of an organism and compare transcriptional changes between organisms/treatments to ascertain specific pathways/genes that are moving in response. But now, let's talk about what can add bias to the data and what we do with the data to make sure that it is reasonable to proceed to further analysis steps.

But first, let's brainstorm a little bit. Look back at the RNA-seq workflow figure above and let's suggest a few places where things could potentially affect the output dataset.

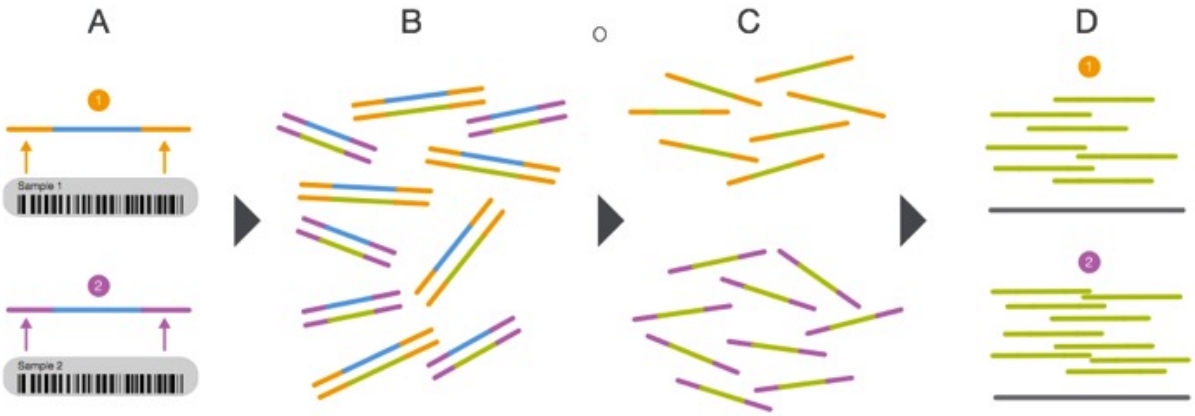
Here are a few thoughts...

- How could the random priming step affect downstream results?
- How could RNA secondary structures affect the library preparation process?
- Would GC content be a problem?
- Could gene length cause issues?
- What might happen if you have genes with substantially different expression levels?
- During the cluster generation on the Illumina flow cell, what might happen if you have too few clusters? Too many?
- How is it possible to sequence many samples at one time?
- What if you run out of reagents from one kit and have to open another kit to finish the library preparation process?
- Could sequencing depth be an issue?

So now that you may be questioning the validity of any RNA-seq dataset, take heart! Many very smart people have thought about these issues and come up with ways to assess technical artifacts and correct for them. So again, let's brainstorm some potential solutions to these problems. Which problems can be addressed through better chemistries/processes vs. mathematical/computational correction?

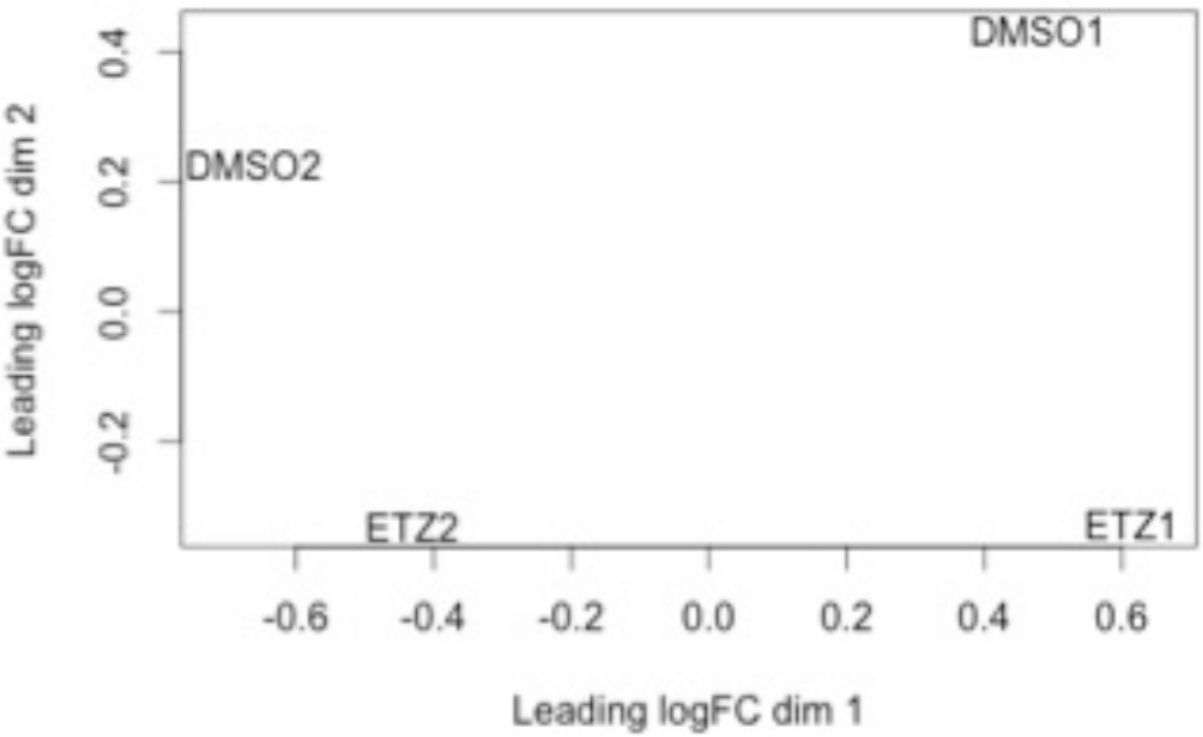
These sorts of issues should always be considered, but recognize that RNA-seq is becoming fairly commonplace and solutions to many of these questions exist. Be critical of your data and *always* look at the raw data.

Multiplexing the sequencing process by pooling several samples together is not only cheaper, it can overcome what are known as *batch effects*. Batch effects are when you have samples that correlate with one another based on batch/time/etc. instead of biological replication. This is a very real phenomenon and can be caused by using different lots of the same kit/flow cells when preparing samples! You can correct for this, but we will get there later... For now, have a look at the diagram showing how multiplexing is achieved.



From: http://www.illumina.com/content/dam/illumina-marketing/documents/products/sequencing_introduction_microbiology.pdf

This is an example of what a *batch effect* looks like. Note how DMSO1 and ETZ1 group together and DMSO2 and ETZ2 group together (e.g. by batch).



We can determine what is considered a "good" base call from a "bad" one through using what is known as the Phred scoring system or Q-score.

Where Q is defined as a property that is logarithmically related to the base call error probability:

$$Q = -10 \log_{10} P \mid error\ probability = P^2$$

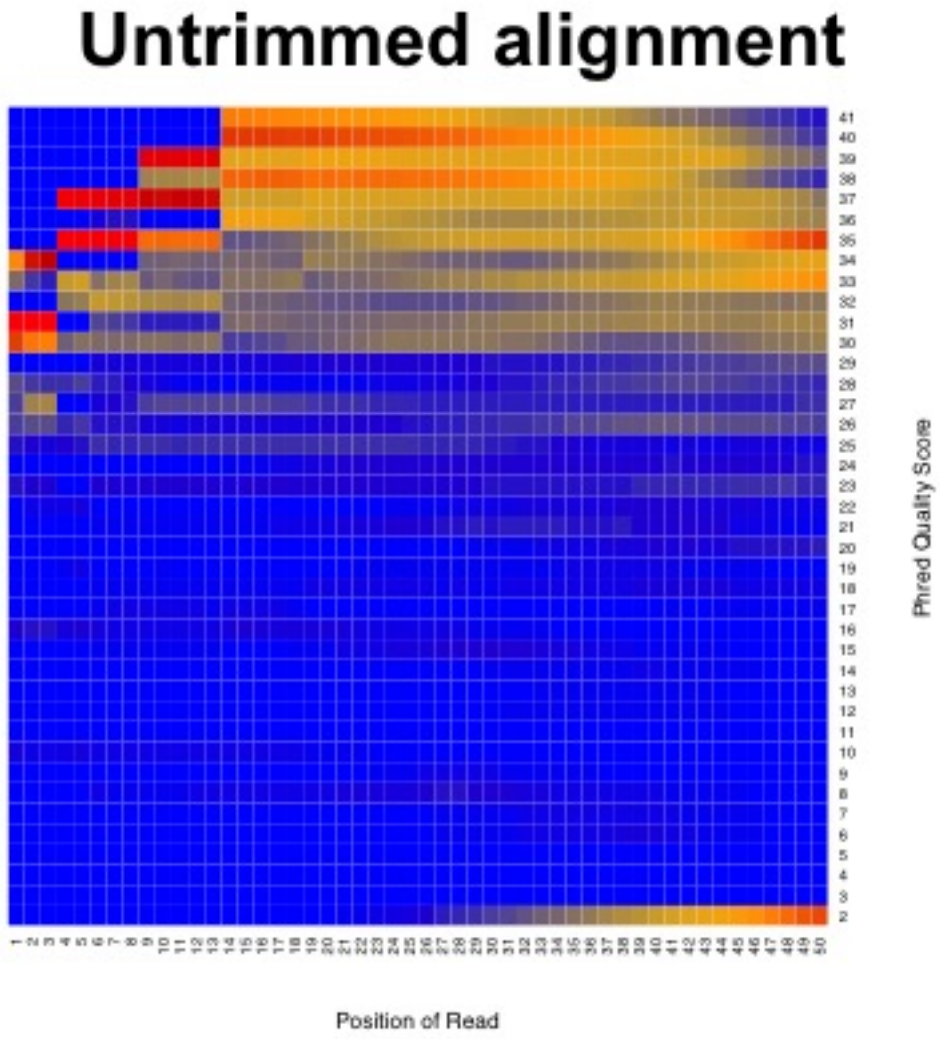
So this means:

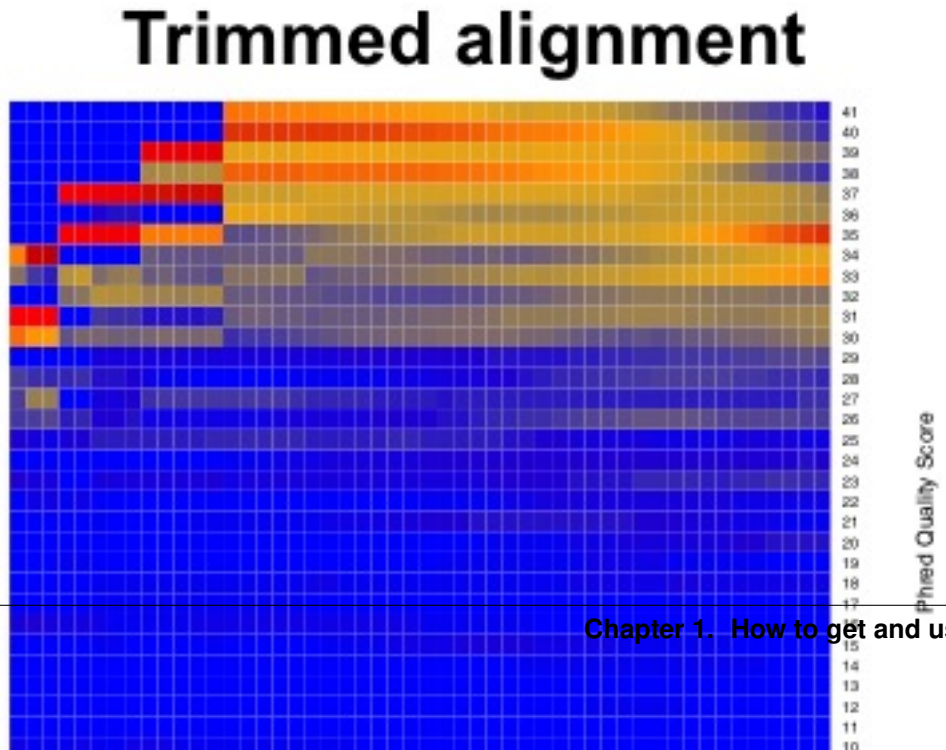
Table 1: Quality Scores and Base Calling Accuracy

Phred Quality Score	Probability of Incorrect Base Call	Base Call Accuracy
10	1 in 10	90%
20	1 in 100	99%
30	1 in 1,000	99.9%
40	1 in 10,000	99.99%
50	1 in 100,000	99.999%

From: http://res.illumina.com/documents/products/technotes/technote_q-scores.pdf

Illumina tends to output sequence results with a Q > 30. So let's have a look at what some raw data looks like in terms of Q-scores before and after trimming adapters and low quality reads.


Untrimmed alignment
Phred Quality Score
Position of Read


Trimmed alignment
Phred Quality Score

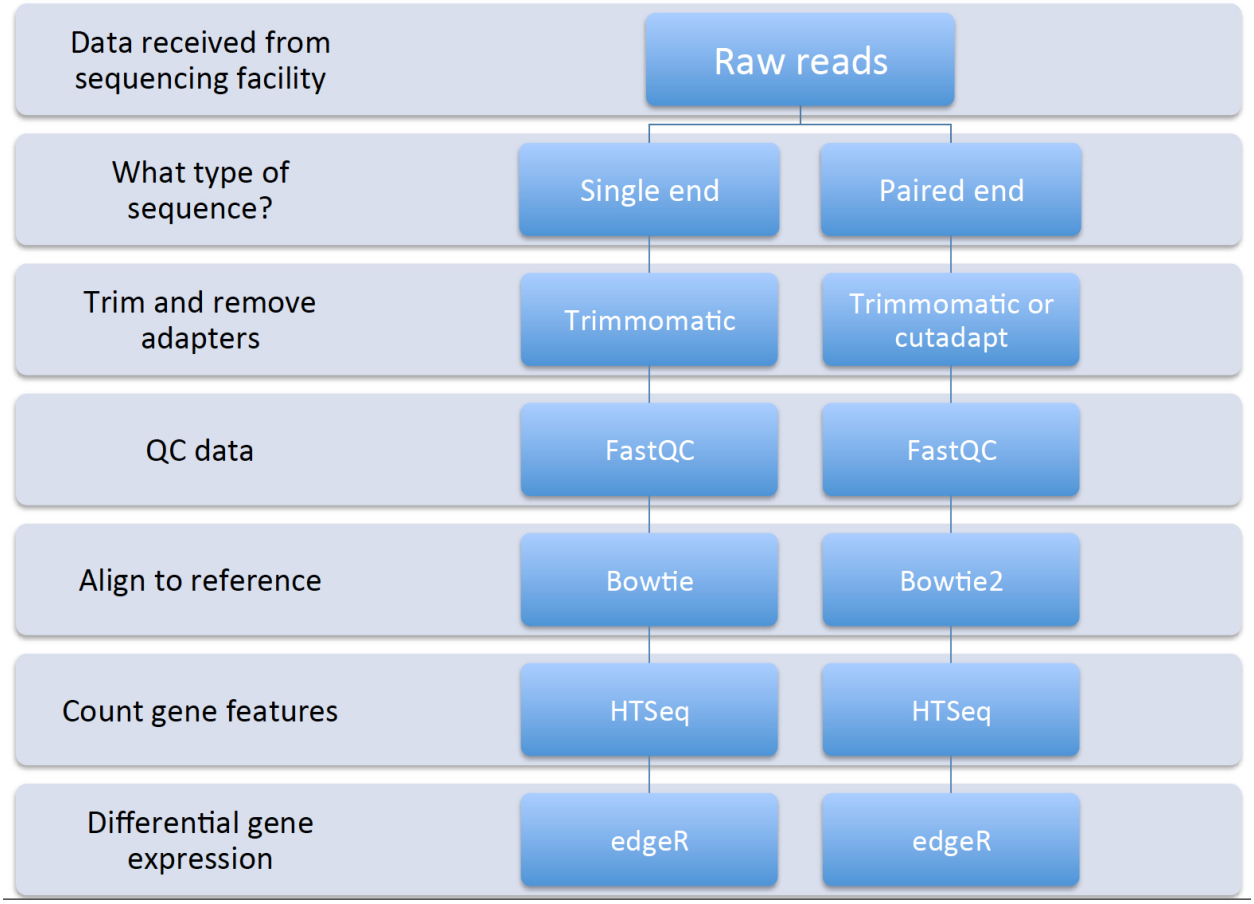
This is why we do the trimming before attempting to align the reads to the reference genome. Since we are using FastQC, let's have a look at some sample data of what good Illumina data looks like.

So, we have come to the end of the background section. Even with all of the great tools and chemistries that have been developed to handle RNA-seq datasets, the old mantra still applies: *garbage in; garbage out* and *with great power comes great responsibility*. Take care in analyzing these sorts of data as they typically influence many downstream experiments.

Questions!

Basic analysis procedure

Now that we have began to understand the background of RNA-seq technologies, how libraries are prepared and sequenced, and thought aboutpotential pitfalls during the data analysis process, let's have a look at the basic workflow and some tools that we will use for each step:



Remember that we can have both single- and paired-end reads. Each type of output will require slightly different tools and procedure. The data that we will be working with is single-end Illumina reads.

Let's brainstorm for a minute:

- If the Illumina sequencing procedure (as seen in the video above) requires specific adapters, what are some ways we could remove them?
- What are some potential issues specifically with our reads that could cause misalignments or no alignments at all to a reference genome?

- Why don't we use a reference transcriptome instead of a genome since RNA-seq is a *transcriptional* profiling experiment?
- What are other genomic features in bacteria that could potentially be identified using RNA-seq data?

Trimmomatic

Trimmomatic is a lightweight java application that can remove Illumina adapter sequences and low quality reads. It uses a sliding window to analyze chunks of each read, examining the quality score, minimum read length, if it corresponds to an adapter sequence, etc. Let's have a look at the documentation to see what each option does.

When we run the analysis, you will likely see some output that looks like this:

```
TrimmomaticSE: Started with arguments: -threads 4 /mnt/home/john3434/RNAseq/Data/gly7a.fq.gz /mnt/hor
Using Long Clipping Sequence: 'AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGTA'
Using Long Clipping Sequence: 'AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC'
ILLUMINACLIP: Using 0 prefix pairs, 2 forward/reverse sequences, 0 forward only sequences, 0 reverse
Quality encoding detected as phred33
Input Reads: 100000 Surviving: 96867 (96.87%) Dropped: 3133 (3.13%)
TrimmomaticSE: Completed successfully
```

Note: It is important to log this output into a text file somewhere and save it (fortunately the software we are going to use will log it for you). You might want this for a report when you're finished.

Let's remind ourselves what each command and parameter is doing. Look through the command and discuss with a neighbor what is going on there. If you don't remember what each parameter does, have another look at the documentation.

Let me know if you have questions by placing a red sticky note on your computer.

FastQC

FastQC is a piece of software that allows us to analyze the quality of our data before proceeding to aligning the reads to the reference genome. Let's have a look again at what good Illumina data and bad Illumina data look like. This will help us determine the quality of our own sequence based on their examples.

The output from FastQC will look like this (with a different file name instead of 'trimmedgly7a.fq.gz':

```
Started analysis of trimmedgly7a.fq.gz
    Approx 5% complete for trimmedgly7a.fq.gz
    Approx 10% complete for trimmedgly7a.fq.gz
    Approx 15% complete for trimmedgly7a.fq.gz
    Approx 20% complete for trimmedgly7a.fq.gz
    Approx 25% complete for trimmedgly7a.fq.gz
    Approx 30% complete for trimmedgly7a.fq.gz
    Approx 35% complete for trimmedgly7a.fq.gz
    Approx 40% complete for trimmedgly7a.fq.gz
    Approx 45% complete for trimmedgly7a.fq.gz
    Approx 50% complete for trimmedgly7a.fq.gz
    Approx 55% complete for trimmedgly7a.fq.gz
    Approx 60% complete for trimmedgly7a.fq.gz
    Approx 65% complete for trimmedgly7a.fq.gz
    Approx 70% complete for trimmedgly7a.fq.gz
    Approx 75% complete for trimmedgly7a.fq.gz
    Approx 80% complete for trimmedgly7a.fq.gz
    Approx 85% complete for trimmedgly7a.fq.gz
    Approx 90% complete for trimmedgly7a.fq.gz
```

```
Approx 95% complete for trimmedgly7a.fq.gz
Analysis complete for trimmedgly7a.fq.gz
```

We can open the report file in a browser like FireFox. Here are two different reports `report1.html` and `report2.html` What do we think? Good or bad data?

Please work with a neighbor and discuss the FastQC analysis reports. Put a green sticky note on your computer once you have done this and viewed the results in a browser.

Bowtie

What is Bowtie?

"Bowtie is an ultrafast, memory-efficient short read aligner geared toward quickly aligning large sets of short DNA sequences (reads) to large genomes... Bowtie indexes the genome with a Burrows-Wheeler index to keep its memory footprint small..."

What isn't Bowtie?

"Bowtie is not a general-purpose alignment tool like MUMer, BLAST, or Vmatch. Bowtie works best when aligning short reads to large genomes, though it supports arbitrarily small reference sequences (e.g. amplicons) and reads as long as 1024 bases. Bowtie is designed to be extremely fast for sets of short reads where (a) many of the reads have at least one good, valid alignment, (b) many of the reads are relatively high-quality, and (c) the number of alignments reported per read is small (close to 1)."

From: http://bowtie-bio.sourceforge.net/manual.shtml#what-is-bowtie

In order for Bowtie to work, we need to provide it with trimmed reads files and the reference genome in a FASTA format file. This type of file typically ends in .fa or .fasta.

We can acquire our favorite reference genome and feature file (GTF) from the Ensembl website.

Once we get our data from the RTSF, we will download the *L. reuteri* JCM1112 `genome file` and `feature file`. The feature file contains data to inform HTSeq where the start and end of a gene is. This is important as HTSeq produces the number of transcripts per gene identified in a given sample.

HTSeq

This step will take the longest time, computationally, out of the entire workflow.

HTSeq is a powerful Python package for analyzing NGS data. For our purposes, we will be using the counting feature of HTSeq. Let's have a look at the way HTSeq can count whether a read maps to a gene.

We need to supply htseq-count with a couple things:

1. A genome feature file (GTF) so that HTSeq "knows" where the start and end of a gene is
2. The *.sam* file that was output from Bowtie

Differential gene expression with edgeR

Up to this point we have done several things: trimmed, QC'd, aligned, and counted reads that mapped to each gene. Now, we will finally move to the step where we will analyze the differential gene expression between the untreated and treated *L. reuteri* samples!

To do this, we have chosen to utilize an analysis package written in the R programming language called edgeR. edgeR stands for differential expression analysis of digital gene expression data in R. This is a fantastic tool that is actively maintained (as seen by the date of the most recent user guide update) and fairly easy to use. Several diagnostic plots are

produced throughout the analysis that provide meaningful information as to whether we can even perform differential gene expression between samples and if there are batch effects we have to deal with.

RNA-seq data does not typically assume a normal (Gaussian) distribution, so to glean which genes are changing in a statistically significant manner, we have to model the data slightly differently. EdgeR implements what is called a negative binomial distribution, sometimes referred to as a gamma-Poisson model. If you *really* enjoy statistics and would like to dig into the mathematical underpinnings of this software, see the references at the bottom of this page. If you are less interested in understanding the math behind all of this, here is the short summary: we need to examine the data to make sure they separate enough between treatments to determine differential gene expression and we *always* use a false-discovery rate correction to determine significance (even then, it's worth looking at the fold-change differences to decide if it is "real"; though this is slightly more arbitrary).

Presentation time!

Please have one person from each treatment group come and present a *representative* report from each treatment, assessing the results.

Note: Save your report so that we can compile them at the end of the module.

1.1.2 Frequently Asked Questions

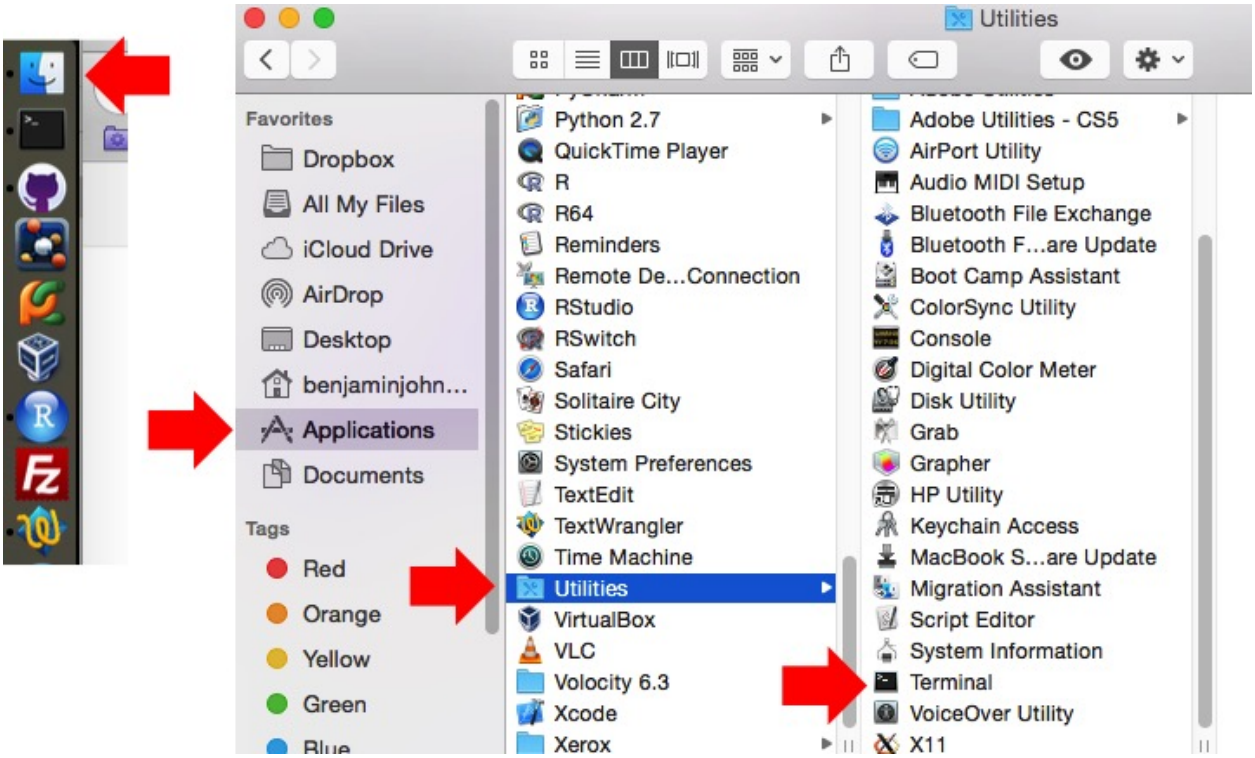
1.1.3 Mac OS X tutorial

Download the workflow: SPARTA for Mac

1. *Introduction*
2. *Basic Terminal Commands*
3. *Install Dependencies*
4. *Initializing SPARTA*
5. *Analyzing Example Data*
6. *Analyzing Your Data*
7. *Identifying Potential Batch Effects*
8. *Altering Workflow Execution Options*

Introduction

Many bioinformatics software packages and workflows require the user to utilize them from the command line or terminal. SPARTA is no different. The reason the command line interface is utilized is that a great deal of power and flexibility can be gained without the use of a graphical user interface (GUI). Further, a GUI can be difficult to implement across various platforms. To find the command line interface/Terminal on Mac OS X, go to Finder -> Applications -> Utilities -> Terminal (might just be worth dragging it onto your dock).



Decompress the SPARTA_Mac-master.zip file by double-clicking on it. Now, drag and drop the decompressed folder onto your desktop.

SPARTA expects either compressed (.gz) or uncompressed FASTQ files (.fq or .fastq) as input, with a reference genome file in FASTA format and a genome feature file (.gtf) within the folder that contains the input data. To see an example of appropriate input data, look inside the ExampleData folder within the SPARTA_Mac-master folder.

To download a reference genome and genome feature file for your favorite bacteria, go to the Ensembl website.

Basic Terminal Commands

Let's have a look at some basic Terminal commands, we will cover the commands necessary to:

1. Move through folders

2. List the contents of a folder

3. Make new folders

4. Rename files/folders

5. Delete files/folders

	Command	What it does	Examples
1.	cd	Change directory/folder	cd ~ (this changes to your home directory); cd .. (this goes back one folder)
2.	ls	List the contents of a folder	ls
3.	mkdir	Make a new directory/folder	mkdir NewFolder (this will make a new folder called 'NewFolder' in your current directory)
4.	mv	Rename or move a file from one name to another	mv file1 file2 (this will rename/move file1 to file2)
5.	rm	Remove a file (add the -r flag to remove a folder)	rm file1 (remove file1); rm -r folder1 (remove folder1)

Command reference sheet

Unix/Linux Command Reference

FOSSwire.com

File Commands

ls – directory listing
ls -al – formatted listing with hidden files
cd *dir* - change directory to *dir*
cd – change to home
pwd – show current directory
mkdir *dir* – create a directory *dir*
rm *file* – delete *file*
rm -r *dir* – delete directory *dir*
rm -f *file* – force remove *file*
rm -rf *dir* – force remove directory *dir* *
cp *file1 file2* – copy *file1* to *file2*
cp -r *dir1 dir2* – copy *dir1* to *dir2*; create *dir2* if it doesn't exist
mv *file1 file2* – rename or move *file1* to *file2* if *file2* is an existing directory, moves *file1* into directory *file2*
ln -s *file link* – create symbolic link *link* to *file*
touch *file* – create or update *file*
cat > *file* – places standard input into *file*
more *file* – output the contents of *file*
head *file* – output the first 10 lines of *file*
tail *file* – output the last 10 lines of *file*
tail -f *file* – output the contents of *file* as it grows, starting with the last 10 lines

Process Management

ps – display your currently active processes
top – display all running processes
kill *pid* – kill process id *pid*
killall *proc* – kill all processes named *proc* *
bg – lists stopped or background jobs; resume a stopped job in the background
fg – brings the most recent job to foreground
fg *n* – brings job *n* to the foreground

File Permissions

chmod *octal file* – change the permissions of *file* to *octal*, which can be found separately for user, group, and world by adding:

- 4 – read (r)
- 2 – write (w)
- 1 – execute (x)

Examples:
chmod 777 – read, write, execute for all
chmod 755 – rwx for owner, rx for group and world
For more options, see **man chmod**.

SSH

ssh *user@host* – connect to *host* as *user*
ssh -p *port user@host* – connect to *host* on port *port* as *user*
ssh-copy-id *user@host* – add your key to *host* for *user* to enable a keyed or passwordless login

Searching

grep *pattern files* – search for *pattern* in *files*
grep -r *pattern dir* – search recursively for *pattern* in *dir*
command* | grep *pattern – search for *pattern* in the output of *command*
locate *file* – find all instances of *file*

System Info

date – show the current date and time
cal – show this month's calendar
uptime – show current uptime
w – display who is online
whoami – who you are logged in as
finger *user* – display information about *user*
uname -a – show kernel information
cat /proc/cpuinfo – cpu information
cat /proc/meminfo – memory information
man *command* – show the manual for *command*
df – show disk usage
du – show directory space usage
free – show memory and swap usage
whereis *app* – show possible locations of *app*
which *app* – show which *app* will be run by default

Compression

tar cf *file.tar files* – create a tar named *file.tar* containing *files*
tar xf *file.tar* – extract the files from *file.tar*
tar czf *file.tar.gz files* – create a tar with Gzip compression
tar xzf *file.tar.gz* – extract a tar using Gzip
tar cjf *file.tar.bz2* – create a tar with Bzip2 compression
tar xjf *file.tar.bz2* – extract a tar using Bzip2
gzip *file* – compresses *file* and renames it to *file.gz*
gzip -d *file.gz* – decompresses *file.gz* back to *file*

Network

ping *host* – ping *host* and output results
whois *domain* – get whois information for *domain*
dig *domain* – get DNS information for *domain*
dig -x *host* – reverse lookup *host*
wget *file* – download *file*
wget -c *file* – continue a stopped download

Installation

Install from source:
./configure
make
make install
dpkg -i *pkg.deb* – install a package (Debian)
rpm -Uvh *pkg.rpm* – install a package (RPM)

Shortcuts

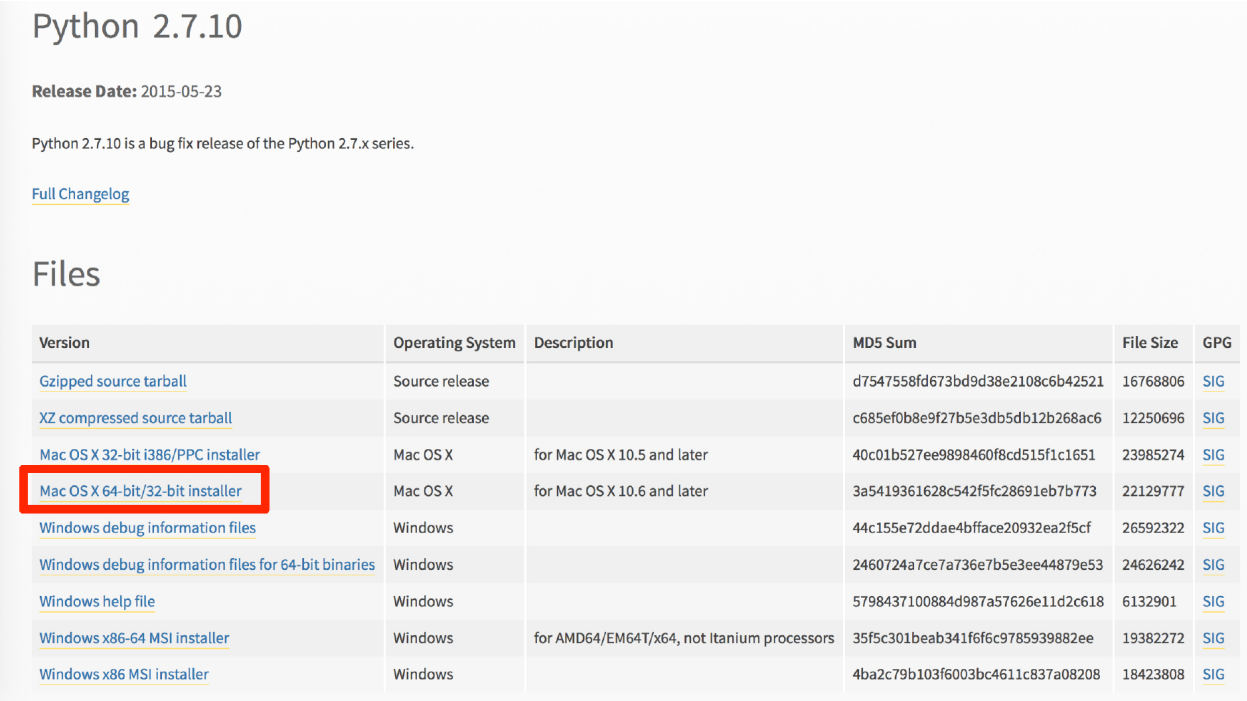
Ctrl+C – halts the current command
Ctrl+Z – stops the current command, resume with **fg** in the foreground or **bg** in the background
Ctrl+D – log out of current session, similar to **exit**
Ctrl+W – erases one word in the current line
Ctrl+U – erases the whole line
Ctrl+R – type to bring up a recent command
!! - repeats the last command
exit – log out of current session

* use with extreme caution.

Ref. sheet from: http://files.fosswire.com/2007/08/fwunixref.pdf

Install Dependencies

The SPARTA workflow requires a few things in order to run: Python, Java, NumPy, and R. If you already have these installed, great! If you don't, let's start by downloading the latest version of Python 2 (see image below). You will want to download and install the red boxed version of Python 2. Follow the prompts to install Python with the default values.



Python 2.7.10

Release Date: 2015-05-23

Python 2.7.10 is a bug fix release of the Python 2.7.x series.

Full Changelog

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		d7547558fd673bd9d38e2108c6b42521	16768806	SIG
XZ compressed source tarball	Source release		c685ef0b8e9f27b5e3db5db12b268ac6	12250696	SIG
Mac OS X 32-bit i386/PPC installer	Mac OS X	for Mac OS X 10.5 and later	40c01b527ee9898460f8cd515f1c1651	23985274	SIG
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	3a5419361628c542f5fc28691eb7b773	22129777	SIG
Windows debug information files	Windows		44c155e72ddae4bfface20932ea2f5cf	26592322	SIG
Windows debug information files for 64-bit binaries	Windows		2460724a7ce7a736e7b5e3ee44879e53	24626242	SIG
Windows help file	Windows		5798437100884d987a57626e11d2c618	6132901	SIG
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64, not Itanium processors	35f5c301beab341f6f6c9785939882ee	19382272	SIG
Windows x86 MSI installer	Windows		4ba2c79b103f6003bc4611c837a08208	18423808	SIG

Great! Let's check and see if Java is already installed on your system. Open up the terminal, (if you don't remember how to do this, head back to the *Introduction*) and type:

```
java -version
```

If Java is already installed, it will produce some output that looks like this:

```
java version "1.8.0_31"
Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07, mixed mode)
```

If the output does *not* look something like this, Java is likely not installed and two of the tools require Java to function (Trimmomatic and FastQC). Let's download and install a suitable version of Java (see image below). You will want to download and install the red boxed version of Java JRE. Follow the prompts to install Java.

To install NumPy, go back to or open the Terminal and type:

```
sudo pip install numpy
```

This will prompt you for your password. Enter your password and hit Enter/Return.

Note: As you type in your password, **no characters will appear** but you *are* entering characters.

Once you have entered your password and hit Enter/Return, NumPy will be downloaded and installed on your system.

Finally, let's install R. Navigate to the SPARTA_Mac folder and go to the folder labeled "Install_R". Within this folder is an R installer. Double-click on the installer and follow the prompts to install R.

Congratulations! You've installed the necessary dependencies to run SPARTA!

Initializing SPARTA

Once SPARTA is initialized, the workflow will seek to identify that all of the necessary dependencies are met. If they are not satisfied, a message specific to what is not installed will appear as output in the terminal window.

To initialize SPARTA, go to the Terminal and navigate to the SPARTA_Mac-master folder on your desktop by typing:

```
cd ~/Desktop/SPARTA_Mac-master
```

To start the workflow, type:

```
python SPARTA.py
```

This will start the software and check for dependencies.

Analyzing Example Data

SPARTA is distributed with some example data. Specifically, it is the first 100,000 reads of each sample from Baker et al..

To begin the analysis, navigate into the SPARTA_Mac-master folder and drag and drop the folder called "Example-Data" out onto the desktop.

If you haven't already, *initialize SPARTA* from the Terminal.

If all the *dependencies* are met, SPARTA will pause and prompt the user:

```
Is the RNAseq data in a folder on the Desktop? (Y or N):
```

Type:

```
Y
```

Hit Enter/Return

Note: SPARTA assumes the data is located in a folder on the desktop by default. It is easiest if all future analyses have the data in a folder (WITHOUT SPACES IN THE NAME) on the desktop.

Now it will prompt the user for the name of the folder:

```
What is the name of the folder on the Desktop containing the RNAseq data?:
```

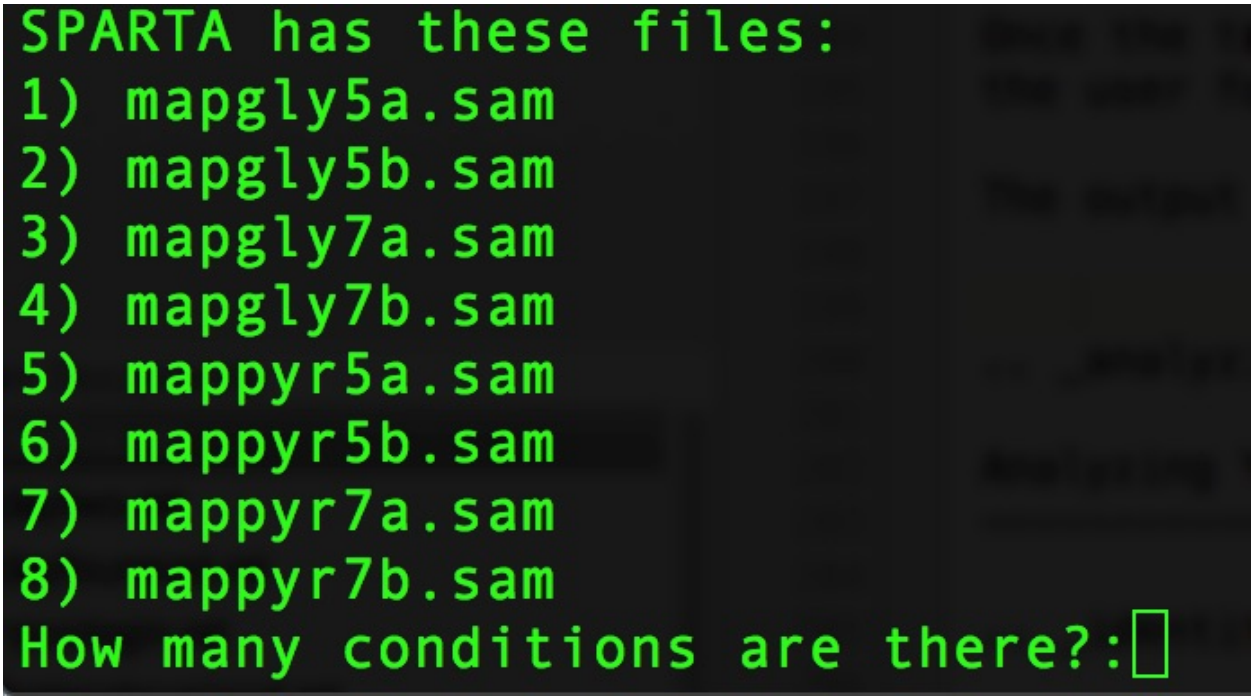
Type:

```
ExampleData
```

This is the name of the folder on the desktop that contains the input example data. Hit Enter/Return. From here, the software will trim, QC, align, and count transcript abundance for each sample. All output/analyses are put in a folder that SPARTA generates on the desktop called "RNAseq_Data". Within this folder are separate folders for each SPARTA run that are denoted by the date (e.g. 2015-06-04). Within these folders are four more folders that separate each step of the analysis and are called: 1) QC, 2) Bowtie, 3) HTSeq, and 4) DEanalysis.

Once the trimming, QC, alignment, and counting are complete, SPARTA will again pause and prompt the user for how many experimental conditions exist within the analysis.

The output at this point will look like this:



At the prompt that says:

```
How many conditions are there?:
```

Type:

```
4
```

Hit Enter/Return. There are 4 experimental conditions that we are considering:

1. Glycerol pH 7.0
2. Glycerol pH 5.7
3. Pyruvate pH 7.0
4. Pyruvate pH 5.7

Each condition has 2 replicates. The next prompt will read:

```
Enter the relevant file names, based on the names given in 'SPARTA has these files', with the replica
As an example, please see the 'conditions_input_example.txt' in the DEanalysis folder.
Once you have entered the file names, hit Enter/Return:
```

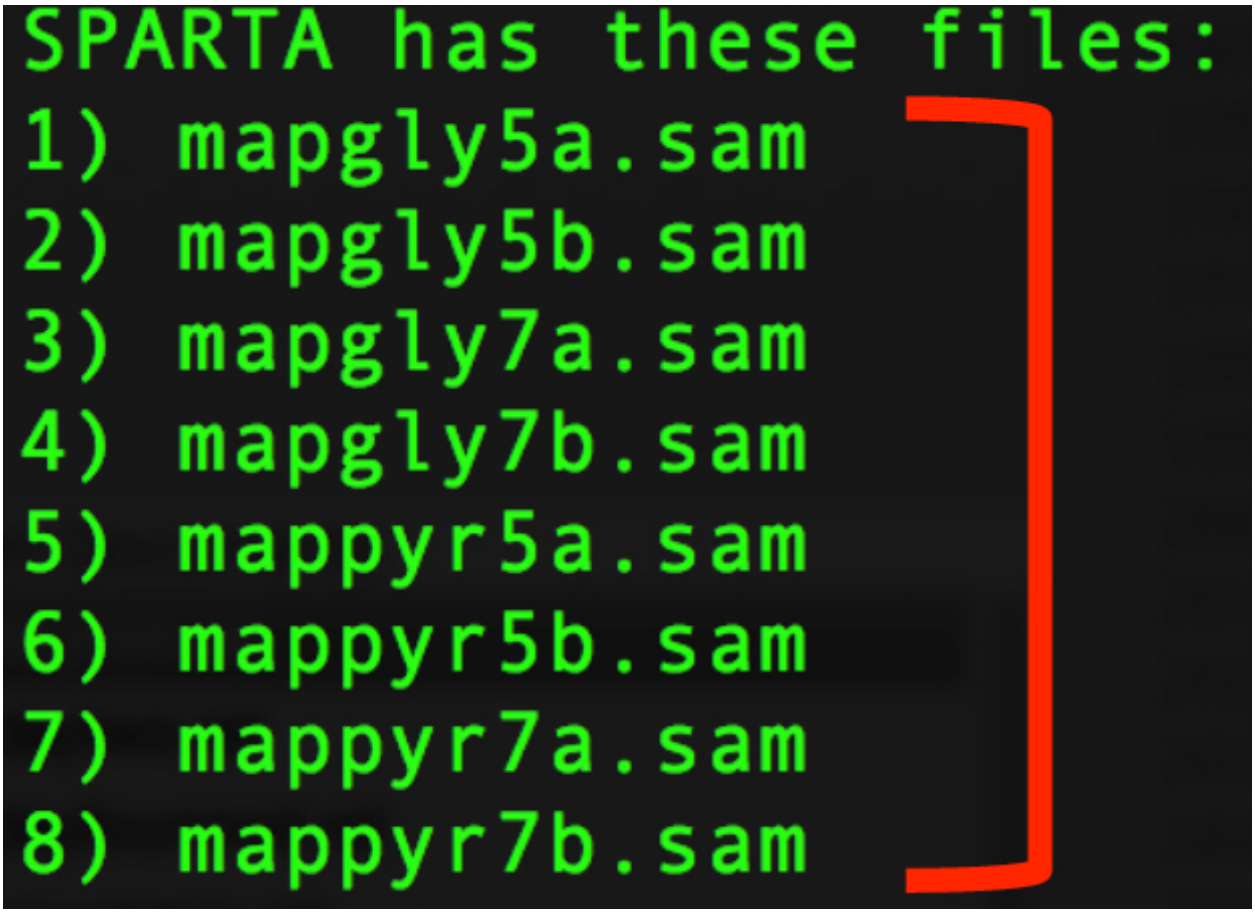
At this point, we need to do a few things.

1. Navigate to the SPARTA output folder called RNAseq_Data located on the desktop
2. Go to the current run folder (will be the last folder listed if sorted by name)
3. Go into the DEanalysis folder
4. Open the conditions_input.txt file in a text editor (NOT MICROSOFT WORD) such as TextEdit

The number of experimental conditions listed are based on the number entered at the prompt asking "How many conditions are there?:". Thus, in our case, there are 4. The contents of the file will look like:

```
Reference_Condition_Files:
Experimental_Condition_2_Files:
Experimental_Condition_3_Files:
Experimental_Condition_4_Files:
```

We now need to enter the file names of the replicates in each condition. These are comma-separated file names that correspond to the output given by SPARTA (denoted with red bracket)



Note: The file names are case-sensitive and must be spelled *exactly* as listed in the output given by SPARTA

Thus, when all the file names are inputed, the conditions_input.txt file should look like this:

```
Reference_Condition_Files: mapgly7a.sam, mapgly7b.sam
Experimental_Condition_2_Files:mapgly5a.sam, mapgly5b.sam
Experimental_Condition_3_Files:mappyr7a.sam, mappyr7b.sam
Experimental_Condition_4_Files:mappyr5a.sam, mappyr5b.sam
```

Now, save the changes by going to File -> Save. Go back to the terminal and hit Enter/Return. From here, the workflow will perform the differential gene expression analysis through edgeR. If a batch effect may be present, the output will attempt to warn the user of the potential, unintended variable that *must* be accounted for before drawing experimental conclusions.

All the differential gene expression output is located in the RNAseq_Data -> date of your current run -> DEanalysis folder. The file output includes:

1. Differential gene expression tables

2. MDS plot (somewhat analogous to a principle component analysis plot) which will show whether your replicates group together and treatment groups separate based on the treatment
3. BCV plot (biological coefficient of variation) to look at gene level variation between samples

Congratulations! You've analyzed RNA-seq data from raw reads to differential gene expression!

Analyzing Your Data

If you haven't already, we recommend working through the *example data analysis* first before attempting to work through your own data set to familiarize yourself with the workflow.

As stated in the *Introduction*, SPARTA expects either compressed (.gz) or uncompressed FASTQ files (.fq or .fastq) as input, with a reference genome file in FASTA format and a genome feature file (.gtf) within the folder that contains the input data on your desktop. To see an example of appropriate input data, look inside the ExampleData folder within the SPARTA_Mac-master folder.

Now, to analyze your own data, follow the steps to *initialize SPARTA*, and start the analysis!

If you would like to tweak the analysis options for a given step/tool, have a look at the *Altering Workflow Execution Options*.

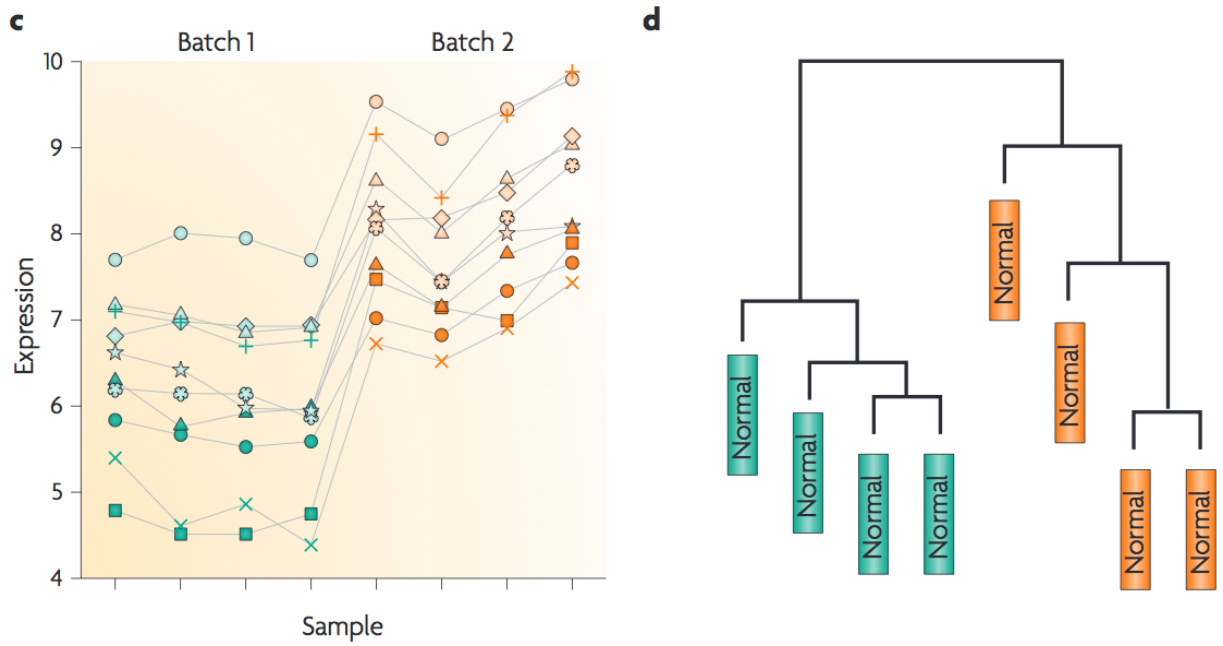
Identifying Potential Batch Effects

Batch effects can be a source of variation in RNA-seq data that can confound biological conclusions. In fact, there have been documented cases of batch effects present in published studies that led readers to be concerned for the validity of the results.

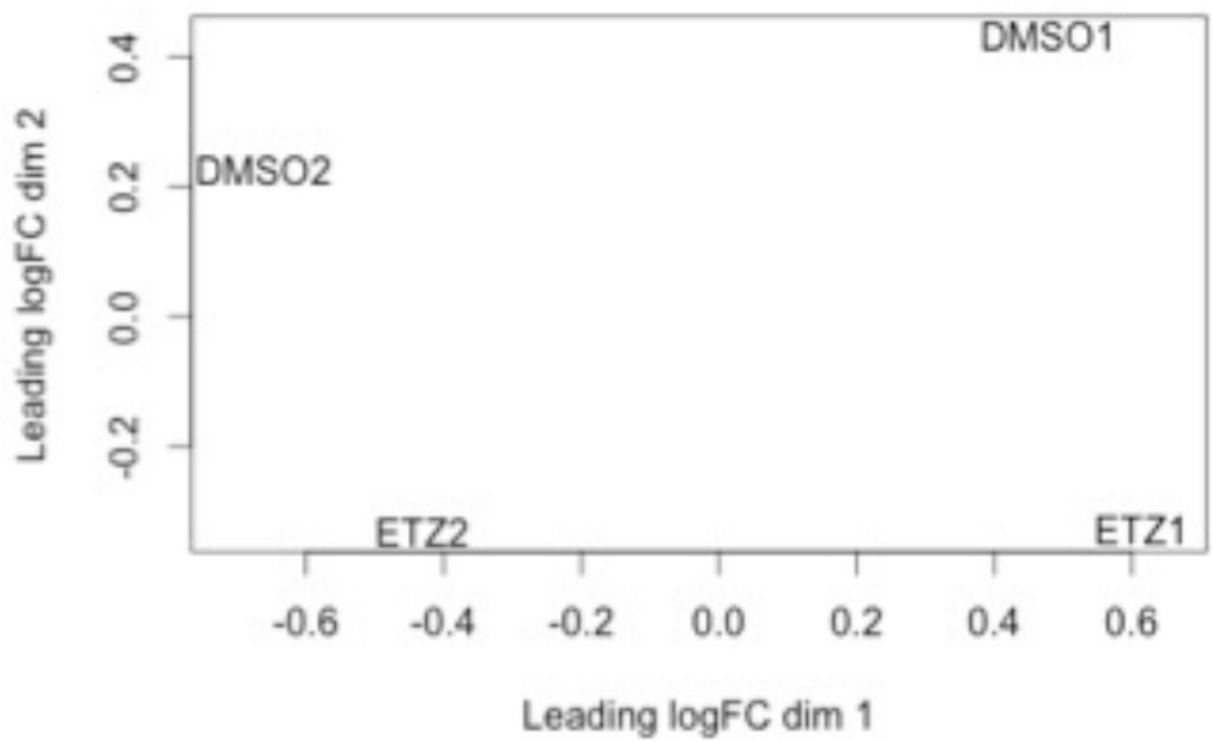
To quote a previously published paper in Nature Reviews Genetics, "Batch effects are sub-groups of measurements that have qualitatively different behaviour across conditions and are unrelated to the biological or scientific variables in a study. For example, batch effects may occur if a subset of experiments was run on Monday and another set on Tuesday, if two technicians were responsible for different subsets of the experiments or if two different lots of reagents, chips or instruments were used."

Thus, it is paramount that one address batch effects within their data before drawing biological conclusions from a specific RNA-seq experiment. To illustrate what a batch effect may look like within the data, we will utilize several different plots.

This first plot comes from the Nature Reviews Genetics paper where they examine Affymetrix data from a published bladder cancer study. You can quickly see that panels C and D from Figure 1 show that samples from batch 1 (blue) cluster together based on gene expression and samples from batch 2 (orange) cluster together.



Within RNA-seq data, using SPARTA and the MDS plot generated by edgeR, another example of batch effects within a study comparing *Mycobacterium tuberculosis* treated with a compound, we can clearly see that the mock-treated samples (DMSO) and compound-treated samples (ETZ) separate based on batch (A vs B) instead of by treatment. Ideally, we would have the samples group together based on treatment as opposed to batch.



If a potential batch effect is detected in the data set, SPARTA will output a message into the terminal that says:

```
IMPORTANT! YOU MAY HAVE A BATCH EFFECT! PLEASE LOOK AT THE MDS PLOT!
```

If this occurs, have a look at the MDS plot in the RNAseq_Data folder -> date of current run -> DEanalysis folder -> MDSplot.png

From here, you will want to adjust your model to account for the batch effect. Within edgeR, this can be accomplished through an additive linear model. The documentation for edgeR contains a tutorial on how to deal with batch effects that can be found here.

Future implementations of SPARTA will include the ability to adjust for batch effects.

Altering Workflow Execution Options

SPARTA is capable of allowing the user to alter the parameters associated with each analysis step to be tailored to specific use cases. Below are the different parameters that can be altered and their usage.

Options:

```
-h, --help            show this help message and exit
--SE                  Single-end read input. Default input choice is single-
                      end if nothing is specified
--PE                  Paired-end read input. Must have the exact same file
                      name and end with _F for the forward read and _R for
                      the reverse read
--cleanup=CLEANUP     Clean up the intermediate files to save space. Default
                      action is to retain the intermediate files. Usage:
                      --cleanup=True
--verbose             Display more output for each step of the analysis.
--noninteractive      Non-interactive mode. This is for running SPARTA
                      without any user input. Assumes data is on the
                      desktop. If this option is specified, you must fill
                      out the configuration file (ConfigFile.txt) with the
                      appropriate experimental conditions in the SPARTA
                      folder.

Trimmomatic options:
  The order the options will be run are: ILLUMINACLIP, LEADING,
  TRAILING, SLIDINGWINDOW, MINLEN

  --clip=ILLUMINACLIP
                      ILLUMINACLIP options. MiSeq & HiSeq usually
                      TruSeq3.fa; GAII usually TruSeq2.fa. Default is
                      ILLUMINACLIP:TruSeq3-SE.fa:2:30:10. Usage:
                      --clip=<adapterseqs>:<seed mismatches>:<palindrome
                      clip threshold>:<simple clip threshold>
  --lead=LEADING      Set the minimun quality required to keep a base.
                      Default is LEADING=3. Usage: --lead=<quality>
  --trail=TRAILING    Set the minimum quality required to keep a base.
                      Default is TRAILING=3. Usage: --trail=<quality>
  --slidewin=SLIDINGWINDOW
                      SLIDINGWINDOW options. Default is SLIDINGWINDOW:4:15.
                      Usage: --slidewin=<window_size>:<required_quality>

HTSeq options:
  --stranded=STRANDED
                      Stranded options: yes, no, reverse. Default is
                      --stranded=reverse. Usage: --stranded=yes/no/reverse
  --order=ORDER       Order options: name, pos. Usage: --order=name/pos.
```

```
  --minqual=MINQUAL    Skip all reads with quality lower than the given
                       value. Default is --minqual=10. Usage:
                       --minqual=<value>
  --idattr=IDATTR      Feature ID from the GTF file to identify counts in the
                       output table Default is --idattr=gene_id. Usage:
                       --idattr=<id attribute>
  --mode=MODE          Mode to handle reads overlapping more than one
                       feature. Default is --mode=union. Usage: --mode=union
                       /intersection-strict/intersection-nonempty
```

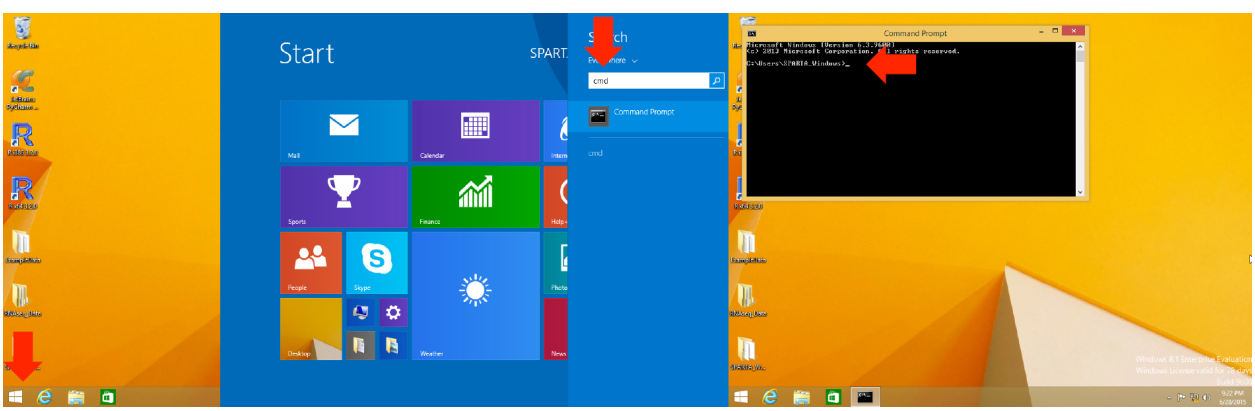
1.1.4 Windows tutorial

Download the workflow: SPARTA for Windows

1. *Introduction*
2. *Basic Terminal Commands*
3. *Install Dependencies*
4. *Initializing SPARTA*
5. *Analyzing Example Data*
6. *Analyzing Your Data*
7. *Identifying Potential Batch Effects*
8. *Altering Workflow Execution Options*

Introduction

Many bioinformatics software packages and workflows require the user to utilize them from the command line or terminal. SPARTA is no different. The reason the command line interface is utilized is that a great deal of power and flexibility can be gained without the use of a graphical user interface (GUI). Further, a GUI can be difficult to implement across various platforms. To find the command line interface/Terminal on Windows, go to Windows start button -> Search -> Type in: cmd -> Terminal is now open to enter commands.



Decompress the SPARTA_Windows-master.zip file by double-clicking on it. Now, drag and drop the decompressed folder onto your desktop.

SPARTA expects either compressed (.gz) or uncompressed FASTQ files (.fq or .fastq) as input, with a reference genome file in FASTA format and a genome feature file (.gtf) within the folder that contains the input data. To see an example of appropriate input data, look inside the ExampleData folder within the SPARTA_Windows-master folder.

To download a reference genome and genome feature file for your favorite bacteria, go to the Ensembl website.

Basic Terminal Commands

Let's have a look at some basic Terminal commands, we will cover the commands necessary to:

1. Move through folders

2. List the contents of a folder

3. Make new folders

4. Rename files/folders

5. Delete files/folders

	Command	What it does	Examples
1.	cd	Change directory/folder	cd ~ (this changes to your home directory); cd .. (this goes back one folder)
2.	dir	List the contents of a folder	dir
3.	mkdir	Make a new directory/folder	mkdir NewFolder (this will make a new folder called 'NewFolder' in your current directory)
4.	move	Rename or move a file from one name to another	move file1 file2 (this will rename/move file1 to file2)
5.	rm	Remove a file (rmdir is the command to remove a folder)	rm file1 (remove file1); rmdir folder1 (remove folder1)

Basic Command Prompt Commands:

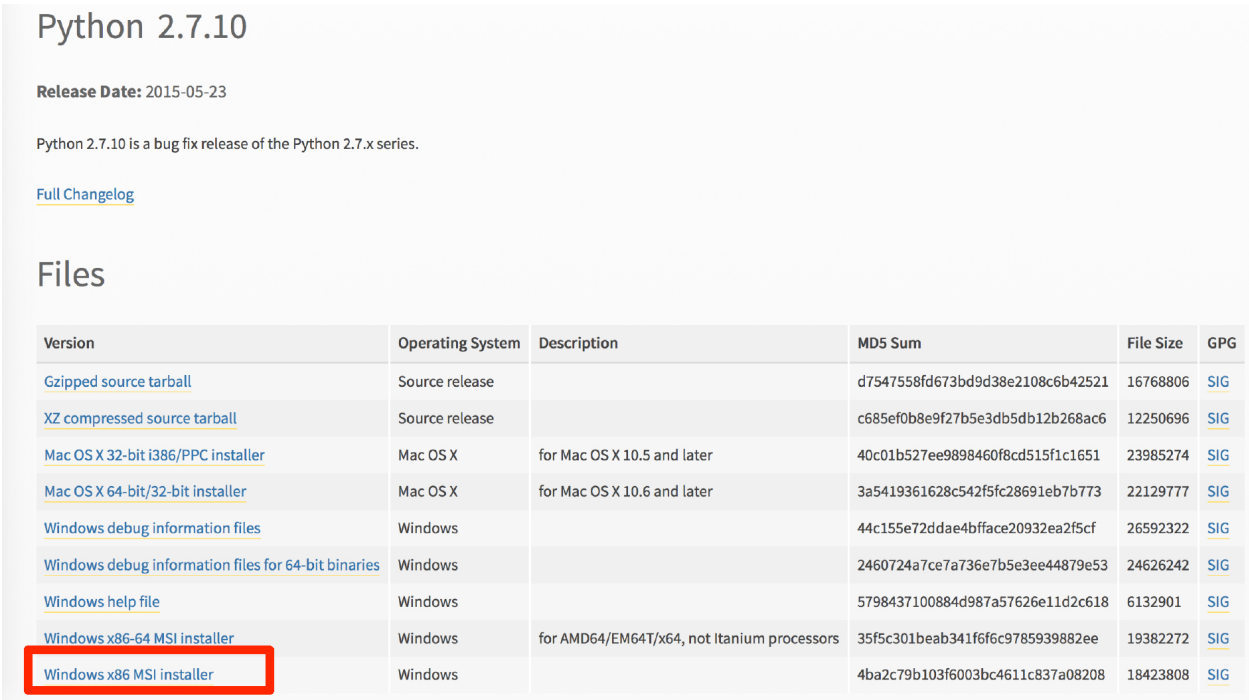
```
x /? = provides syntax info and complete list of all parameters for x (a command, like "cd")
cd = change directory
cd .. = move to the parent directory
cd\ = move to the root of current drive
cd x = move to the current\x directory
cd z: = change to the z root directory (as opposed to c:\)
copy x y = copy file x to directory y (Ex: D:\games\galaga.exe C:\programs[\awesome.exe]), [] = optio
copy file con = display file contents in console
copy con file.txt = create text file in the console window, end with ctrl+z (^z or F6)
date = change the date
del = delete/erase
del x = deletes all files/folders fitting x
del . = deletes all files within current directory
del *.* = deletes all files within current directory
dir = display contents of current directory (Ex: dir [c:][\programs]), [] = optional
dir *.txt = list all .txt files in current directory
dir *.? = list all files with extensions one character in length in current directory
dir /w /p *.* = display all contents one screen at a time
dir | more = display all contents one line at a time
dir /? = provides syntax info and complete list of all dir parameters
echo = send command line input to display (by default)
echo sometext >> somefile.txt = append line(s) of text to any file
echo sometext > somefile.txt = overwrites file with sometext
erase = delete/erase
exit = exit the command prompt
filename.txt = opens filename.txt in current directory in Notepad (or default .txt program)
format z: = format z drive [Ex: use to format a disc or flash drive]
mkdir x = make directory x in current directory
move x y = more or rename x to y
```

```
q = escapes sequential display of contents (i.e. the more parameter)
rd x = remove/delete directory x if it's empty
ren x y = rename file x to y
time = change the time
type file = display the contents of the file 'file' (displays file contents in console)
type file |more = display the contents one line at a time
```

Ref. sheet from: http://blog.simplyadvanced.net/cheat-sheet-for-windows-command-prompt/

Install Dependencies

The SPARTA workflow requires a few things in order to run: Python, Java, NumPy, and R. If you already have these installed, great! If you don't, let's start by downloading the latest version of Python 2 (see image below). You will want to download and install the red boxed version of Python 2. Follow the prompts to install Python with the default values.



Python 2.7.10

Release Date: 2015-05-23

Python 2.7.10 is a bug fix release of the Python 2.7.x series.

Full Changelog

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		d7547558fd673bd9d38e2108c6b42521	16768806	SIG
XZ compressed source tarball	Source release		c685ef0b8e9f27b5e3db5db12b268ac6	12250696	SIG
Mac OS X 32-bit i386/PPC installer	Mac OS X	for Mac OS X 10.5 and later	40c01b527ee9898460f8cd515f1c1651	23985274	SIG
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	3a5419361628c542f5fc28691eb7b773	22129777	SIG
Windows debug information files	Windows		44c155e72ddae4bfface20932ea2f5cf	26592322	SIG
Windows debug information files for 64-bit binaries	Windows		2460724a7ce7a736e7b5e3ee44879e53	24626242	SIG
Windows help file	Windows		5798437100884d987a57626e11d2c618	6132901	SIG
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64, not Itanium processors	35f5c301beab341f6f6c9785939882ee	19382272	SIG
Windows x86 MSI installer	Windows		4ba2c79b103f6003bc4611c837a08208	18423808	SIG

Great! Let's check and see if Java is already installed on your system. Open up the terminal, (if you don't remember how to do this, head back to the *Introduction*) and type:

```
java -version
```

If Java is already installed, it will produce some output that looks like this:

```
java version "1.8.0_31"
Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07, mixed mode)
```

If the output does *not* look something like this, Java is likely not installed and two of the tools require Java to function (Trimmomatic and FastQC). Let's download and install a suitable version of Java (see image below). You will want to download and install the red boxed version of Java JRE. Follow the prompts to install Java.

To install the remaining dependencies, SPARTA is distributed with installers for each remaining piece of software, however, there is an ideal order with which to install them.

Navigate to the SPARTA_Windows-master folder and then into the "Software_To_Install" folder. Inside this folder is a series of executable installers. Double-click and install them in the following order:

1. numpy
2. vcredist
3. HTSeq
4. R
5. gzip

Now, there is one remaining batch file called "add_python_and_R_to_path.bat". This will add the Python, R, and gzip executables to your path so you can run them from the terminal. To execute this script, right-click on the file and then click on the option called "Run as administrator". Windows may warn you that this script is unsafe because it is from an unknown developer. Click on the "Details" button and then click on "Run anyway".

Note: If this script is not run, SPARTA will not function properly.

Congratulations! You've installed the necessary dependencies to run SPARTA!

Initializing SPARTA

Once SPARTA is initialized, the workflow will seek to identify that all of the necessary dependencies are met. If they are not satisfied, a message specific to what is not installed will appear as output in the terminal window.

To initialize SPARTA, go to the Terminal and navigate to the SPARTA_Windows-master folder on your desktop by typing:

```
cd Desktop\SPARTA_Windows-master
```

To start the workflow, type:

```
python SPARTA.py
```

This will start the software and check for dependencies.

Analyzing Example Data

SPARTA is distributed with some example data. Specifically, it is the first 100,000 reads of each sample from Baker et al..

To begin the analysis, navigate into the SPARTA_Mac-master folder and drag and drop the folder called "Example-Data" out onto the desktop.

If you haven't already, *initialize SPARTA* from the Terminal.

If all the *dependencies* are met, SPARTA will pause and prompt the user:

```
Is the RNAseq data in a folder on the Desktop? (Y or N):
```

Type:

```
Y
```

Hit Enter/Return

Note: SPARTA assumes the data is located in a folder on the desktop by default. It is easiest if all future analyses have the data in a folder (WITHOUT SPACES IN THE NAME) on the desktop.

Now it will prompt the user for the name of the folder:

```
What is the name of the folder on the Desktop containing the RNAseq data?:
```

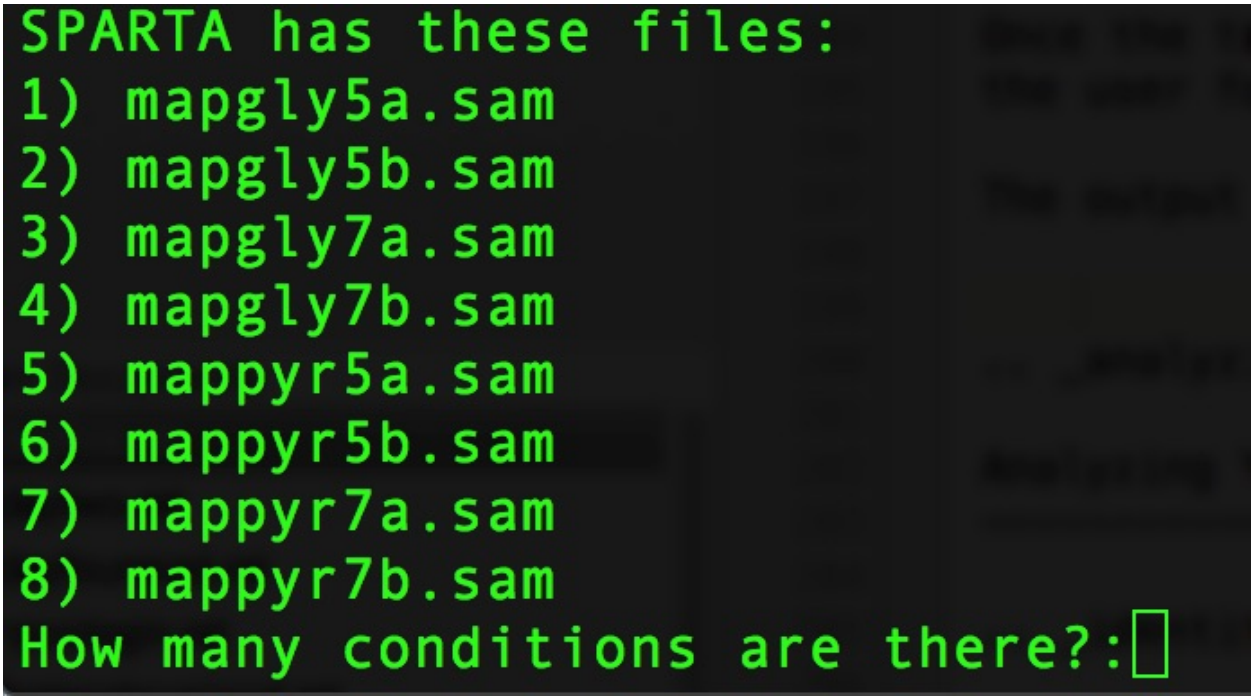
Type:

```
ExampleData
```

This is the name of the folder on the desktop that contains the input example data. Hit Enter/Return. From here, the software will trim, QC, align, and count transcript abundance for each sample. All output/analyses are put in a folder that SPARTA generates on the desktop called "RNAseq_Data". Within this folder are separate folders for each SPARTA run that are denoted by the date (e.g. 2015-06-04). Within these folders are four more folders that separate each step of the analysis and are called: 1) QC, 2) Bowtie, 3) HTSeq, and 4) DEanalysis.

Once the trimming, QC, alignment, and counting are complete, SPARTA will again pause and prompt the user for how many experimental conditions exist within the analysis.

The output at this point will look like this:



At the prompt that says:

```
How many conditions are there?:
```

Type:

```
4
```

Hit Enter/Return. There are 4 experimental conditions that we are considering:

1. Glycerol pH 7.0
2. Glycerol pH 5.7
3. Pyruvate pH 7.0
4. Pyruvate pH 5.7

Each condition has 2 replicates. The next prompt will read:

```
Enter the relevant file names, based on the names given in 'SPARTA has these files', with the replica
As an example, please see the 'conditions_input_example.txt' in the DEanalysis folder.
Once you have entered the file names, hit Enter/Return:
```

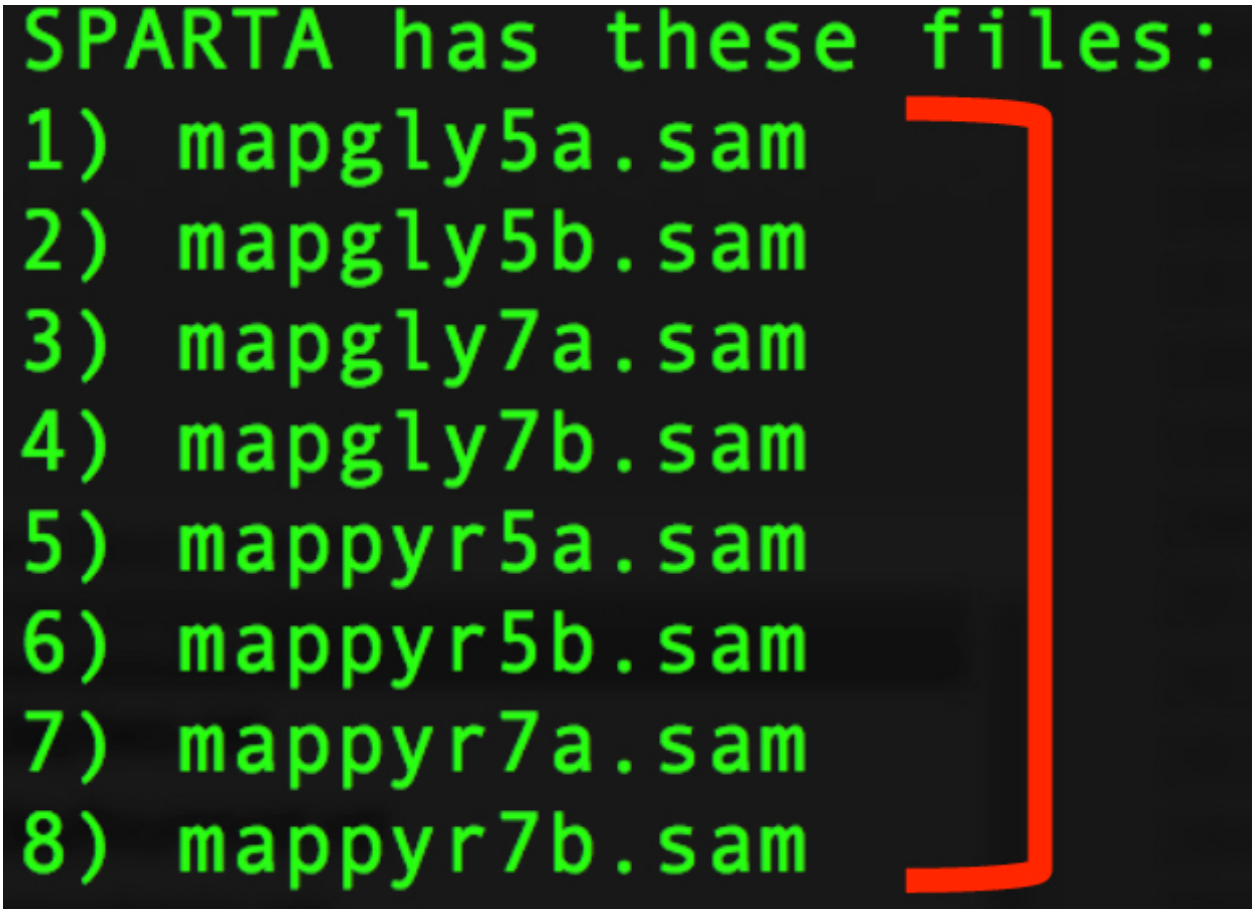
At this point, we need to do a few things.

1. Navigate to the SPARTA output folder called RNAseq_Data located on the desktop
2. Go to the current run folder (will be the last folder listed if sorted by name)
3. Go into the DEanalysis folder
4. Open the conditions_input.txt file in a text editor (NOT MICROSOFT WORD) such as TextEdit

The number of experimental conditions listed are based on the number entered at the prompt asking "How many conditions are there?:". Thus, in our case, there are 4. The contents of the file will look like:

```
Reference_Condition_Files:
Experimental_Condition_2_Files:
Experimental_Condition_3_Files:
Experimental_Condition_4_Files:
```

We now need to enter the file names of the replicates in each condition. These are comma-separated file names that correspond to the output given by SPARTA (denoted with red bracket)



Note: The file names are case-sensitive and must be spelled *exactly* as listed in the output given by SPARTA

Thus, when all the file names are inputed, the conditions_input.txt file should look like this:

```
Reference_Condition_Files: mapgly7a.sam, mapgly7b.sam
Experimental_Condition_2_Files:mapgly5a.sam, mapgly5b.sam
Experimental_Condition_3_Files:mappyr7a.sam, mappyr7b.sam
Experimental_Condition_4_Files:mappyr5a.sam, mappyr5b.sam
```

Now, save the changes by going to File -> Save. Go back to the terminal and hit Enter/Return. From here, the workflow will perform the differential gene expression analysis through edgeR. If a batch effect may be present, the output will attempt to warn the user of the potential, unintended variable that *must* be accounted for before drawing experimental conclusions.

All the differential gene expression output is located in the RNAseq_Data -> date of your current run -> DEanalysis folder. The file output includes:

1. Differential gene expression tables

2. MDS plot (somewhat analogous to a principle component analysis plot) which will show whether your replicates group together and treatment groups separate based on the treatment
3. BCV plot (biological coefficient of variation) to look at gene level variation between samples

Congratulations! You've analyzed RNA-seq data from raw reads to differential gene expression!

Analyzing Your Data

If you haven't already, we recommend working through the *example data analysis* first before attempting to work through your own data set to familiarize yourself with the workflow.

As stated in the *Introduction*, SPARTA expects either compressed (.gz) or uncompressed FASTQ files (.fq or .fastq) as input, with a reference genome file in FASTA format and a genome feature file (.gtf) within the folder that contains the input data on your desktop. To see an example of appropriate input data, look inside the ExampleData folder within the SPARTA_Windows-master folder.

Now, to analyze your own data, follow the steps to *initialize SPARTA*, and start the analysis!

If you would like to tweak the analysis options for a given step/tool, have a look at the *Altering Workflow Execution Options*.

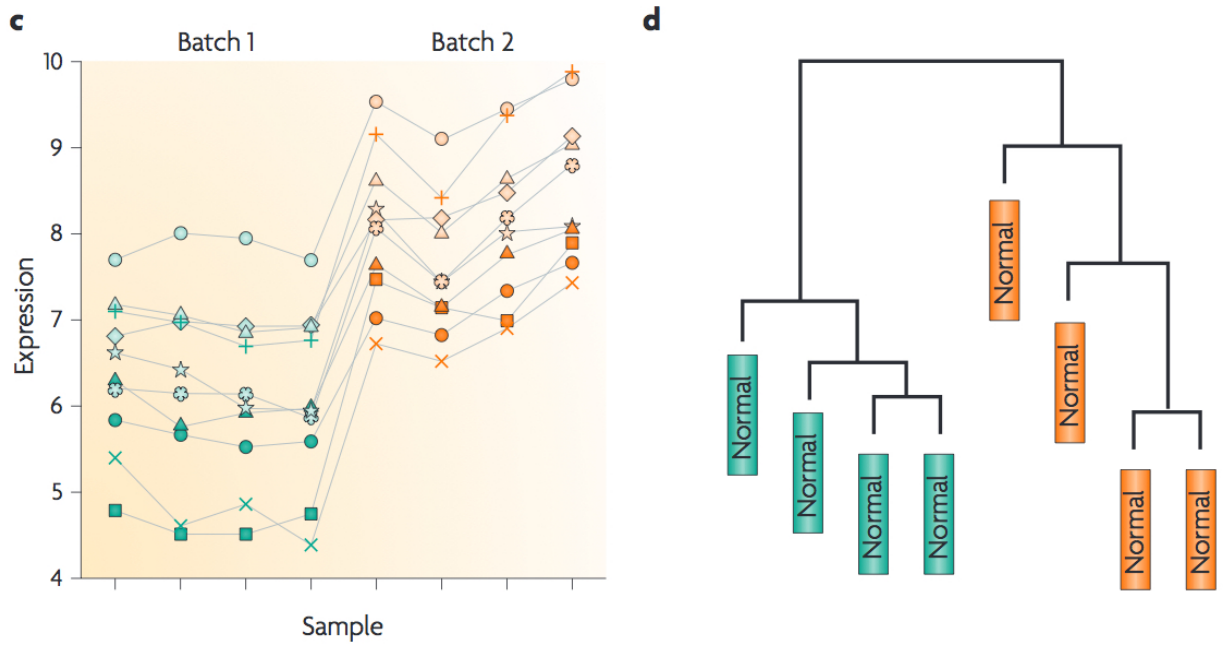
Identifying Potential Batch Effects

Batch effects can be a source of variation in RNA-seq data that can confound biological conclusions. In fact, there have been documented cases of batch effects present in published studies that led readers to be concerned for the validity of the results.

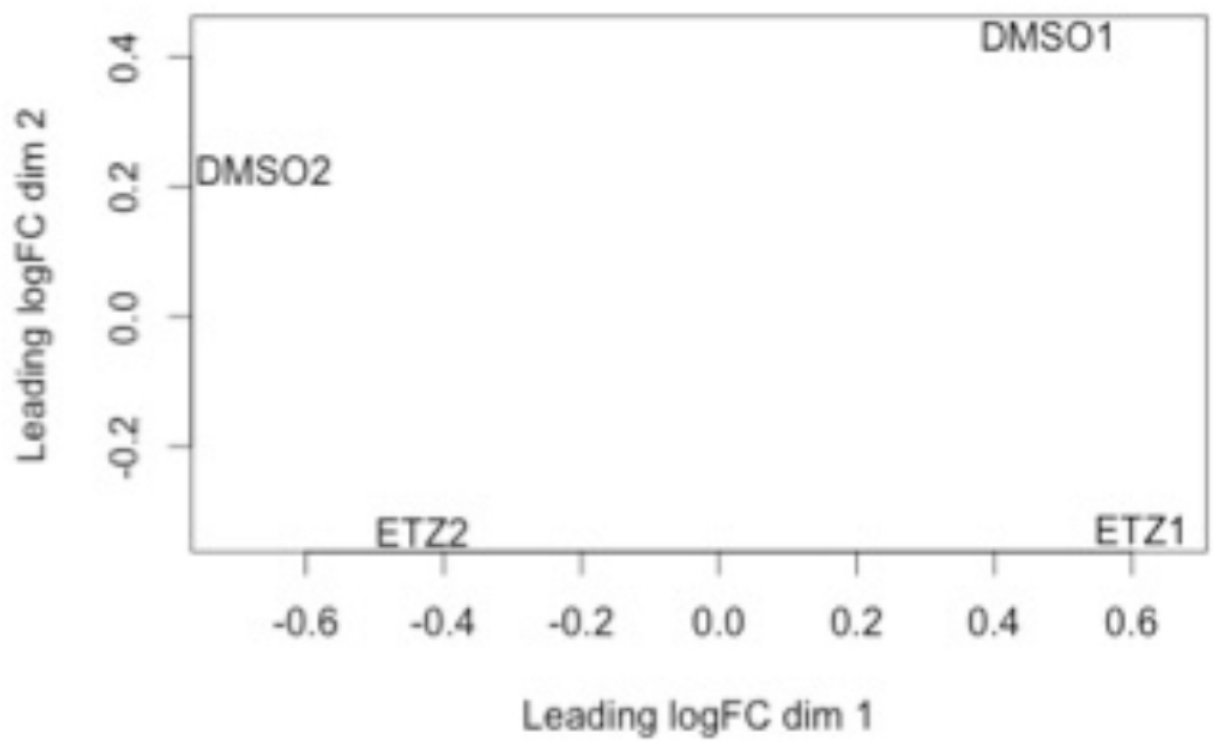
To quote a previously published paper in Nature Reviews Genetics, "Batch effects are sub-groups of measurements that have qualitatively different behaviour across conditions and are unrelated to the biological or scientific variables in a study. For example, batch effects may occur if a subset of experiments was run on Monday and another set on Tuesday, if two technicians were responsible for different subsets of the experiments or if two different lots of reagents, chips or instruments were used."

Thus, it is paramount that one address batch effects within their data before drawing biological conclusions from a specific RNA-seq experiment. To illustrate what a batch effect may look like within the data, we will utilize several different plots.

This first plot comes from the Nature Reviews Genetics paper where they examine Affymetrix data from a published bladder cancer study. You can quickly see that panels C and D from Figure 1 show that samples from batch 1 (blue) cluster together based on gene expression and samples from batch 2 (orange) cluster together.



Within RNA-seq data, using SPARTA and the MDS plot generated by edgeR, another example of batch effects within a study comparing *Mycobacterium tuberculosis* treated with a compound, we can clearly see that the mock-treated samples (DMSO) and compound-treated samples (ETZ) separate based on batch (A vs B) instead of by treatment. Ideally, we would have the samples group together based on treatment as opposed to batch.



If a potential batch effect is detected in the data set, SPARTA will output a message into the terminal that says:

```
IMPORTANT! YOU MAY HAVE A BATCH EFFECT! PLEASE LOOK AT THE MDS PLOT!
```

If this occurs, have a look at the MDS plot in the RNAseq_Data folder -> date of current run -> DEanalysis folder -> MDSplot.png

From here, you will want to adjust your model to account for the batch effect. Within edgeR, this can be accomplished through an additive linear model. The documentation for edgeR contains a tutorial on how to deal with batch effects that can be found here.

Future implementations of SPARTA will include the ability to adjust for batch effects.

Altering Workflow Execution Options

SPARTA is capable of allowing the user to alter the parameters associated with each analysis step to be tailored to specific use cases. Below are the different parameters that can be altered and their usage.

Options:

```
-h, --help            show this help message and exit
--cleanup=CLEANUP     Clean up the intermediate files to save space. Default
                      action is to retain the intermediate files. Usage:
                      --cleanup=True
--verbose             Display more output for each step of the analysis.
--noninteractive      Non-interactive mode. This is for running SPARTA
                      without any user input. Assumes data is on the
                      desktop. If this option is specified, you must fill
                      out the configuration file (ConfigFile.txt) with the
                      appropriate experimental conditions in the SPARTA
                      folder.

Trimmomatic options:
  The order the options will be run are: ILLUMINACLIP, LEADING,
  TRAILING, SLIDINGWINDOW, MINLEN

  --clip=ILLUMINACLIP
                      ILLUMINACLIP options. MiSeq & HiSeq usually
                      TruSeq3.fa; GAII usually TruSeq2.fa. Default is
                      ILLUMINACLIP:TruSeq3-SE.fa:2:30:10. Usage:
                      --clip=<adapterseqs>:<seed mismatches>:<palindrome
                      clip threshold>:<simple clip threshold>
  --lead=LEADING      Set the minimun quality required to keep a base.
                      Default is LEADING=3. Usage: --lead=<quality>
  --trail=TRAILING    Set the minimum quality required to keep a base.
                      Default is TRAILING=3. Usage: --trail=<quality>
  --slidewin=SLIDINGWINDOW
                      SLIDINGWINDOW options. Default is SLIDINGWINDOW:4:15.
                      Usage: --slidewin=<window_size>:<required_quality>

HTSeq options:
  --stranded=STRANDED
                      Stranded options: yes, no, reverse. Default is
                      --stranded=reverse. Usage: --stranded=yes/no/reverse
  --order=ORDER       Order options: name, pos. Usage: --order=name/pos.
  --minqual=MINQUAL   Skip all reads with quality lower than the given
                      value. Default is --minqual=10. Usage:
                      --minqual=<value>
  --idattr=IDATTR     Feature ID from the GTF file to identify counts in the
                      output table Default is --idattr=gene_id. Usage:
```

```
                        --idattr=<id attribute>
  --mode=MODE           Mode to handle reads overlapping more than one
                        feature. Default is --mode=union. Usage: --mode=union
                        /intersection-strict/intersection-nonempty
```

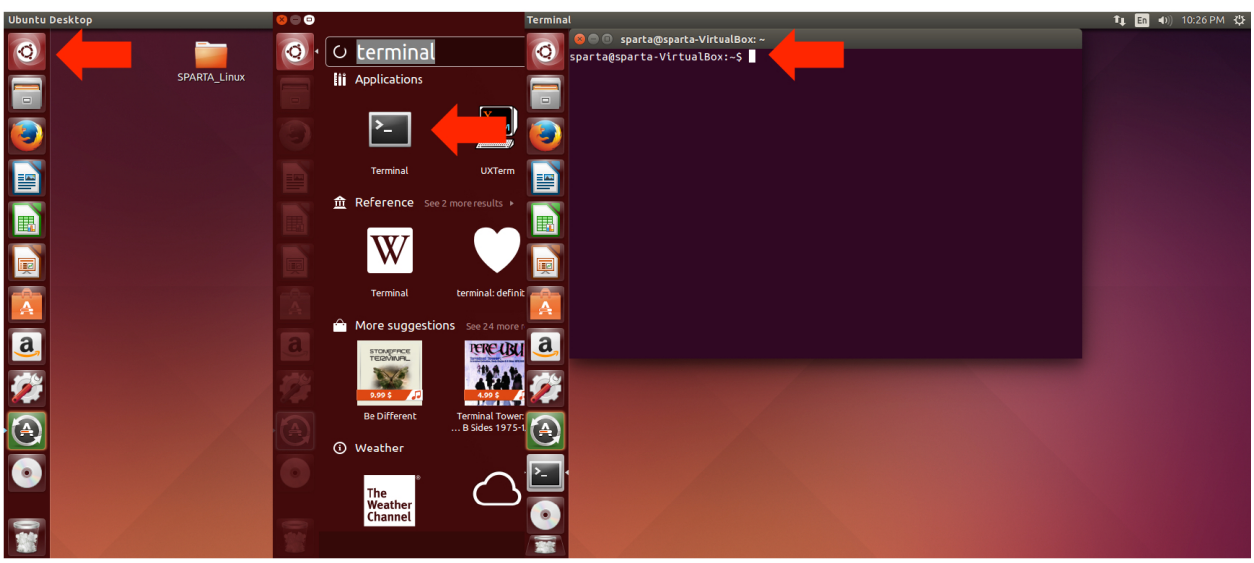
1.1.5 Linux tutorial

Download the workflow: SPARTA for Linux

1. *Introduction*
2. *Basic Terminal Commands*
3. *Install Dependencies*
4. *Initializing SPARTA*
5. *Analyzing Example Data*
6. *Analyzing Your Data*
7. *Identifying Potential Batch Effects*
8. *Altering Workflow Execution Options*

Introduction

Many bioinformatics software packages and workflows require the user to utilize them from the command line or terminal. SPARTA is no different. The reason the command line interface is utilized is that a great deal of power and flexibility can be gained without the use of a graphical user interface (GUI). Further, a GUI can be difficult to implement across various platforms. To find the command line interface/Terminal on Linux (shown in Ubuntu with red arrows), go to "Search your computer and online sources" button -> Search for "terminal" -> Click on Terminal -> Terminal is now open and ready to enter commands (might just be worth dragging it onto your dock).



Decompress the SPARTA_Linux-master.zip file by clicking on it and extracting all the files to the desktop.

SPARTA expects either compressed (.gz) or uncompressed FASTQ files (.fq or .fastq) as input, with a reference genome file in FASTA format and a genome feature file (.gtf) within the folder that contains the input data. To see an example of appropriate input data, look inside the ExampleData folder within the SPARTA_Linux-master folder.

To download a reference genome and genome feature file for your favorite bacteria, go to the Ensembl website.

Basic Terminal Commands

Let's have a look at some basic Terminal commands, we will cover the commands necessary to:

1. Move through folders

2. List the contents of a folder

3. Make new folders

4. Rename files/folders

5. Delete files/folders

	Command	What it does	Examples
1.	cd	Change directory/folder	cd ~ (this changes to your home directory); cd .. (this goes back one folder)
2.	ls	List the contents of a folder	ls
3.	mkdir	Make a new directory/folder	mkdir NewFolder (this will make a new folder called 'NewFolder' in your current directory)
4.	mv	Rename or move a file from one name to another	mv file1 file2 (this will rename/move file1 to file2)
5.	rm	Remove a file (add the -r flag to remove a folder)	rm file1 (remove file1); rm -r folder1 (remove folder1)

Command reference sheet

Unix/Linux Command Reference

FOSSwire.com

File Commands

ls – directory listing
ls -al – formatted listing with hidden files
cd *dir* - change directory to *dir*
cd – change to home
pwd – show current directory
mkdir *dir* – create a directory *dir*
rm *file* – delete *file*
rm -r *dir* – delete directory *dir*
rm -f *file* – force remove *file*
rm -rf *dir* – force remove directory *dir* *
cp *file1 file2* – copy *file1* to *file2*
cp -r *dir1 dir2* – copy *dir1* to *dir2*; create *dir2* if it doesn't exist
mv *file1 file2* – rename or move *file1* to *file2* if *file2* is an existing directory, moves *file1* into directory *file2*
ln -s *file link* – create symbolic link *link* to *file*
touch *file* – create or update *file*
cat > *file* – places standard input into *file*
more *file* – output the contents of *file*
head *file* – output the first 10 lines of *file*
tail *file* – output the last 10 lines of *file*
tail -f *file* – output the contents of *file* as it grows, starting with the last 10 lines

Process Management

ps – display your currently active processes
top – display all running processes
kill *pid* – kill process id *pid*
killall *proc* – kill all processes named *proc* *
bg – lists stopped or background jobs; resume a stopped job in the background
fg – brings the most recent job to foreground
fg *n* – brings job *n* to the foreground

File Permissions

chmod *octal file* – change the permissions of *file* to *octal*, which can be found separately for user, group, and world by adding:

- 4 – read (r)
- 2 – write (w)
- 1 – execute (x)

Examples:
chmod 777 – read, write, execute for all
chmod 755 – rwx for owner, rx for group and world
For more options, see **man chmod**.

SSH

ssh *user@host* – connect to *host* as *user*
ssh -p *port user@host* – connect to *host* on port *port* as *user*
ssh-copy-id *user@host* – add your key to *host* for *user* to enable a keyed or passwordless login

Searching

grep *pattern files* – search for *pattern* in *files*
grep -r *pattern dir* – search recursively for *pattern* in *dir*
command* | grep *pattern – search for *pattern* in the output of *command*
locate *file* – find all instances of *file*

System Info

date – show the current date and time
cal – show this month's calendar
uptime – show current uptime
w – display who is online
whoami – who you are logged in as
finger *user* – display information about *user*
uname -a – show kernel information
cat /proc/cpuinfo – cpu information
cat /proc/meminfo – memory information
man *command* – show the manual for *command*
df – show disk usage
du – show directory space usage
free – show memory and swap usage
whereis *app* – show possible locations of *app*
which *app* – show which *app* will be run by default

Compression

tar cf *file.tar files* – create a tar named *file.tar* containing *files*
tar xf *file.tar* – extract the files from *file.tar*
tar czf *file.tar.gz files* – create a tar with Gzip compression
tar xzf *file.tar.gz* – extract a tar using Gzip
tar cjf *file.tar.bz2* – create a tar with Bzip2 compression
tar xjf *file.tar.bz2* – extract a tar using Bzip2
gzip *file* – compresses *file* and renames it to *file.gz*
gzip -d *file.gz* – decompresses *file.gz* back to *file*

Network

ping *host* – ping *host* and output results
whois *domain* – get whois information for *domain*
dig *domain* – get DNS information for *domain*
dig -x *host* – reverse lookup *host*
wget *file* – download *file*
wget -c *file* – continue a stopped download

Installation

Install from source:
./configure
make
make install
dpkg -i *pkg.deb* – install a package (Debian)
rpm -Uvh *pkg.rpm* – install a package (RPM)

Shortcuts

Ctrl+C – halts the current command
Ctrl+Z – stops the current command, resume with **fg** in the foreground or **bg** in the background
Ctrl+D – log out of current session, similar to **exit**
Ctrl+W – erases one word in the current line
Ctrl+U – erases the whole line
Ctrl+R – type to bring up a recent command
!! - repeats the last command
exit – log out of current session

* use with extreme caution.

Ref. sheet from: http://files.fosswire.com/2007/08/fwunixref.pdf

Install Dependencies

The SPARTA workflow requires a few things in order to run: Python, Java, NumPy, and R. If you already have these installed, great! If you don't, let's start by downloading and installing the dependencies by running the bash script called "install_dependencies.sh".

To run this script, navigate to the SPARTA_Linux-master folder on the desktop:

```
cd ~/Desktop/SPARTA_Linux-master
```

Now, type:

```
bash install_dependencies.sh
```

This will update, download, and install the necessary dependencies to run SPARTA.

Congratulations! You've installed the necessary dependencies to run SPARTA!

Initializing SPARTA

Once SPARTA is initialized, the workflow will seek to identify that all of the necessary dependencies are met. If they are not satisfied, a message specific to what is not installed will appear as output in the terminal window.

To initialize SPARTA, go to the Terminal and navigate to the SPARTA_Linux-master folder on your desktop by typing:

```
cd ~/Desktop/SPARTA_Linux-master
```

To start the workflow, type:

```
python SPARTA.py
```

This will start the software and check for dependencies.

Analyzing Example Data

SPARTA is distributed with some example data. Specifically, it is the first 100,000 reads of each sample from Baker et al..

To begin the analysis, navigate into the SPARTA_Linux-master folder and drag and drop the folder called "Example-Data" out onto the desktop.

If you haven't already, *initialize SPARTA* from the Terminal.

If all the *dependencies* are met, SPARTA will pause and prompt the user:

```
Is the RNAseq data in a folder on the Desktop? (Y or N):
```

Type:

```
Y
```

Hit Enter/Return

Note: SPARTA assumes the data is located in a folder on the desktop by default. It is easiest if all future analyses have the data in a folder (WITHOUT SPACES IN THE NAME) on the desktop.

Now it will prompt the user for the name of the folder:

```
What is the name of the folder on the Desktop containing the RNAseq data?:
```

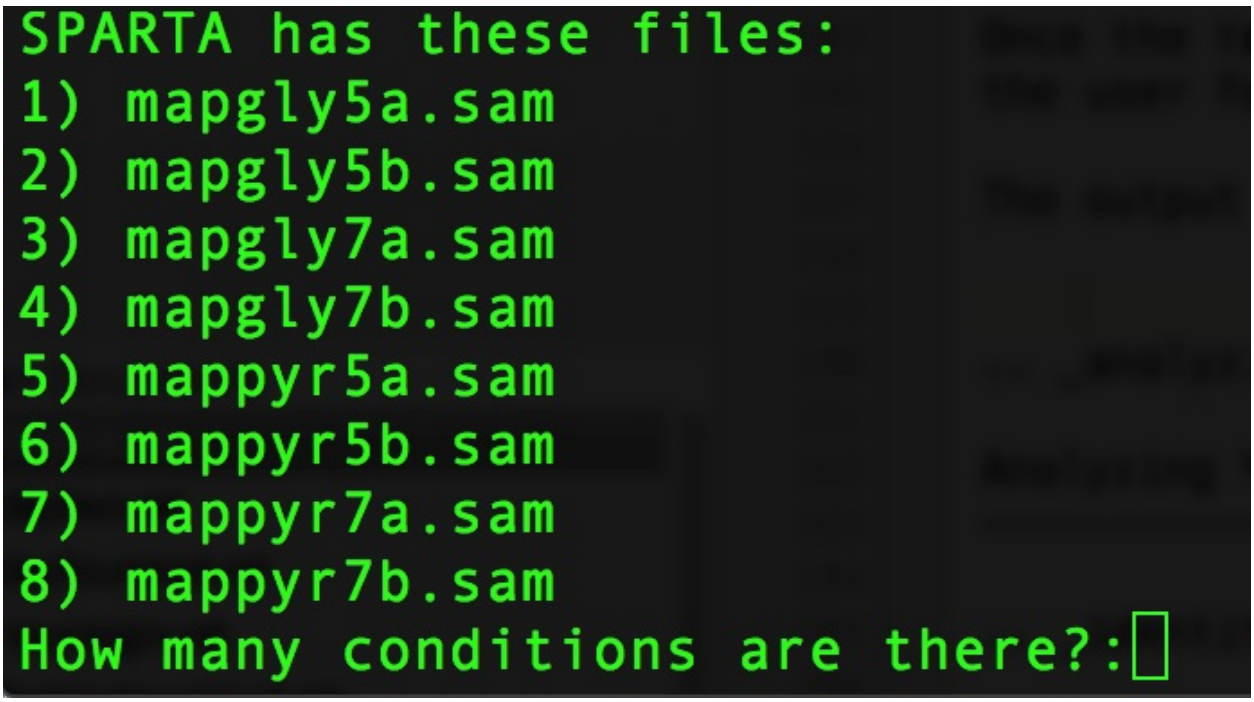
Type:

```
ExampleData
```

This is the name of the folder on the desktop that contains the input example data. Hit Enter/Return. From here, the software will trim, QC, align, and count transcript abundance for each sample. All output/analyses are put in a folder that SPARTA generates on the desktop called "RNAseq_Data". Within this folder are separate folders for each SPARTA run that are denoted by the date (e.g. 2015-06-04). Within these folders are four more folders that separate each step of the analysis and are called: 1) QC, 2) Bowtie, 3) HTSeq, and 4) DEanalysis.

Once the trimming, QC, alignment, and counting are complete, SPARTA will again pause and prompt the user for how many experimental conditions exist within the analysis.

The output at this point will look like this:



At the prompt that says:

```
How many conditions are there?:
```

Type:

```
4
```

Hit Enter/Return. There are 4 experimental conditions that we are considering:

1. Glycerol pH 7.0
2. Glycerol pH 5.7
3. Pyruvate pH 7.0
4. Pyruvate pH 5.7

Each condition has 2 replicates. The next prompt will read:

```
Enter the relevant file names, based on the names given in 'SPARTA has these files', with the replica
As an example, please see the 'conditions_input_example.txt' in the DEanalysis folder.
Once you have entered the file names, hit Enter/Return:
```

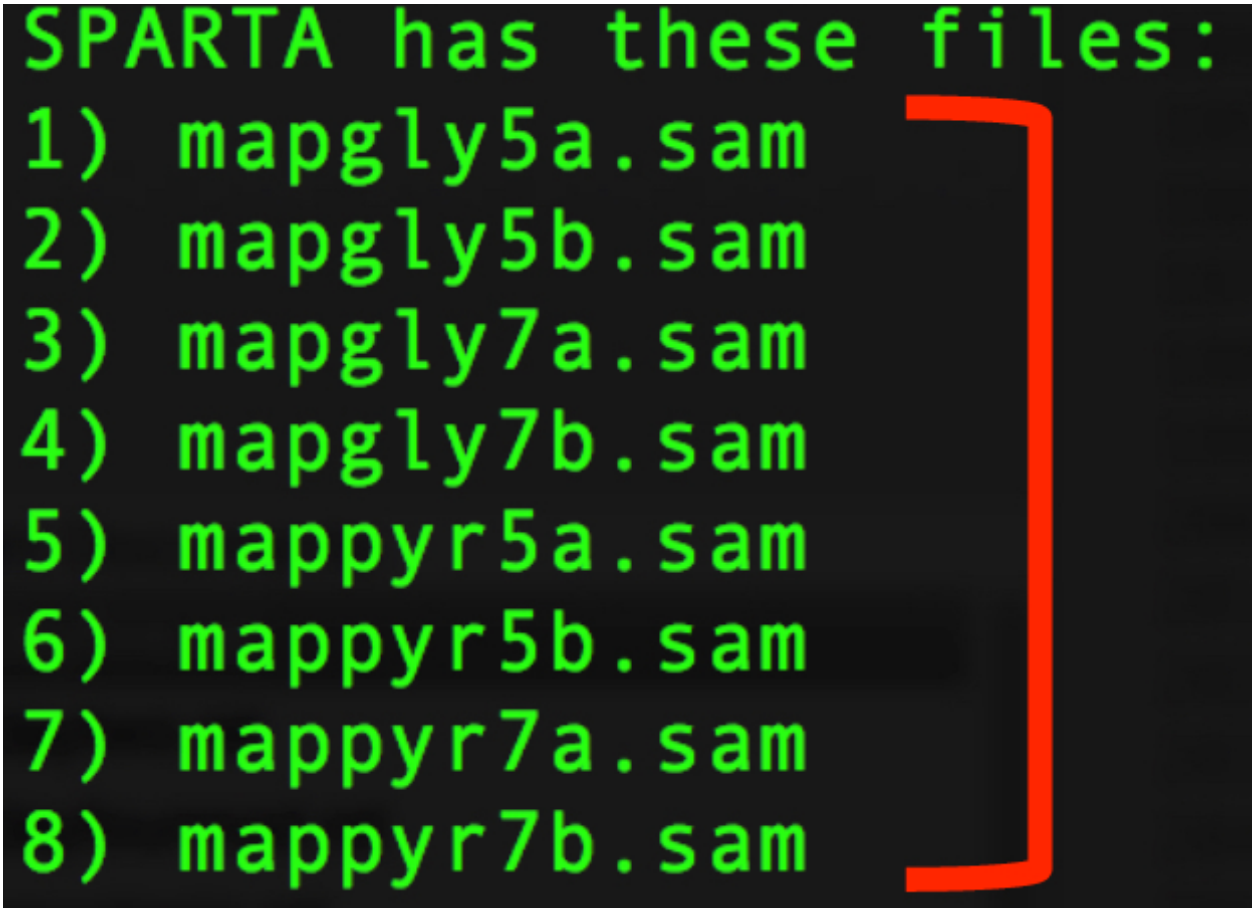
At this point, we need to do a few things.

1. Navigate to the SPARTA output folder called RNAseq_Data located on the desktop
2. Go to the current run folder (will be the last folder listed if sorted by name)
3. Go into the DEanalysis folder
4. Open the conditions_input.txt file in a text editor (NOT MICROSOFT WORD) such as TextEdit

The number of experimental conditions listed are based on the number entered at the prompt asking "How many conditions are there?:". Thus, in our case, there are 4. The contents of the file will look like:

```
Reference_Condition_Files:
Experimental_Condition_2_Files:
Experimental_Condition_3_Files:
Experimental_Condition_4_Files:
```

We now need to enter the file names of the replicates in each condition. These are comma-separated file names that correspond to the output given by SPARTA (denoted with red bracket)



Note: The file names are case-sensitive and must be spelled *exactly* as listed in the output given by SPARTA

Thus, when all the file names are inputed, the conditions_input.txt file should look like this:

```
Reference_Condition_Files: mapgly7a.sam, mapgly7b.sam
Experimental_Condition_2_Files:mapgly5a.sam, mapgly5b.sam
Experimental_Condition_3_Files:mappyr7a.sam, mappyr7b.sam
Experimental_Condition_4_Files:mappyr5a.sam, mappyr5b.sam
```

Now, save the changes by going to File -> Save. Go back to the terminal and hit Enter/Return. From here, the workflow will perform the differential gene expression analysis through edgeR. If a batch effect may be present, the output will attempt to warn the user of the potential, unintended variable that *must* be accounted for before drawing experimental conclusions.

All the differential gene expression output is located in the RNAseq_Data -> date of your current run -> DEanalysis folder. The file output includes:

1. Differential gene expression tables
2. MDS plot (somewhat analogous to a principle component analysis plot) which will show whether your replicates group together and treatment groups separate based on the treatment
3. BCV plot (biological coefficient of variation) to look at gene level variation between samples

Congratulations! You've analyzed RNA-seq data from raw reads to differential gene expression!

Analyzing Your Data

If you haven't already, we recommend working through the *example data analysis* first before attempting to work through your own data set to familiarize yourself with the workflow.

As stated in the *Introduction*, SPARTA expects either compressed (.gz) or uncompressed FASTQ files (.fq or .fastq) as input, with a reference genome file in FASTA format and a genome feature file (.gtf) within the folder that contains the input data on your desktop. To see an example of appropriate input data, look inside the ExampleData folder within the SPARTA_Mac-master folder.

Now, to analyze your own data, follow the steps to *initialize SPARTA*, and start the analysis!

If you would like to tweak the analysis options for a given step/tool, have a look at the *Altering Workflow Execution Options*.

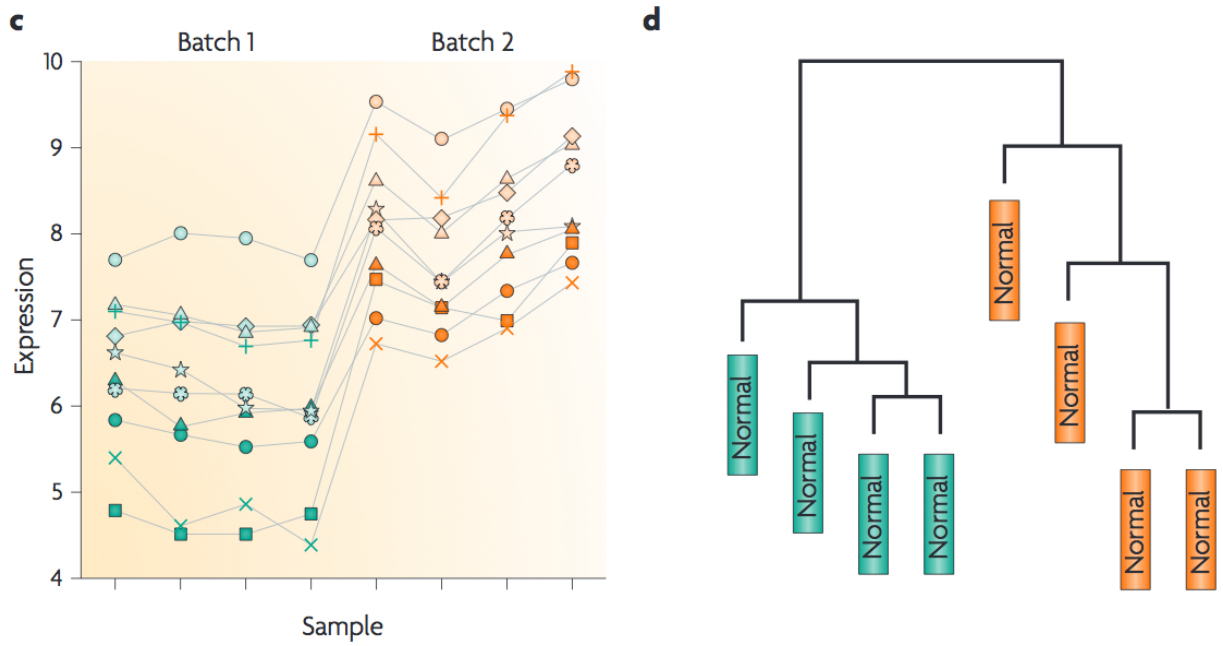
Identifying Potential Batch Effects

Batch effects can be a source of variation in RNA-seq data that can confound biological conclusions. In fact, there have been documented cases of batch effects present in published studies that led readers to be concerned for the validity of the results.

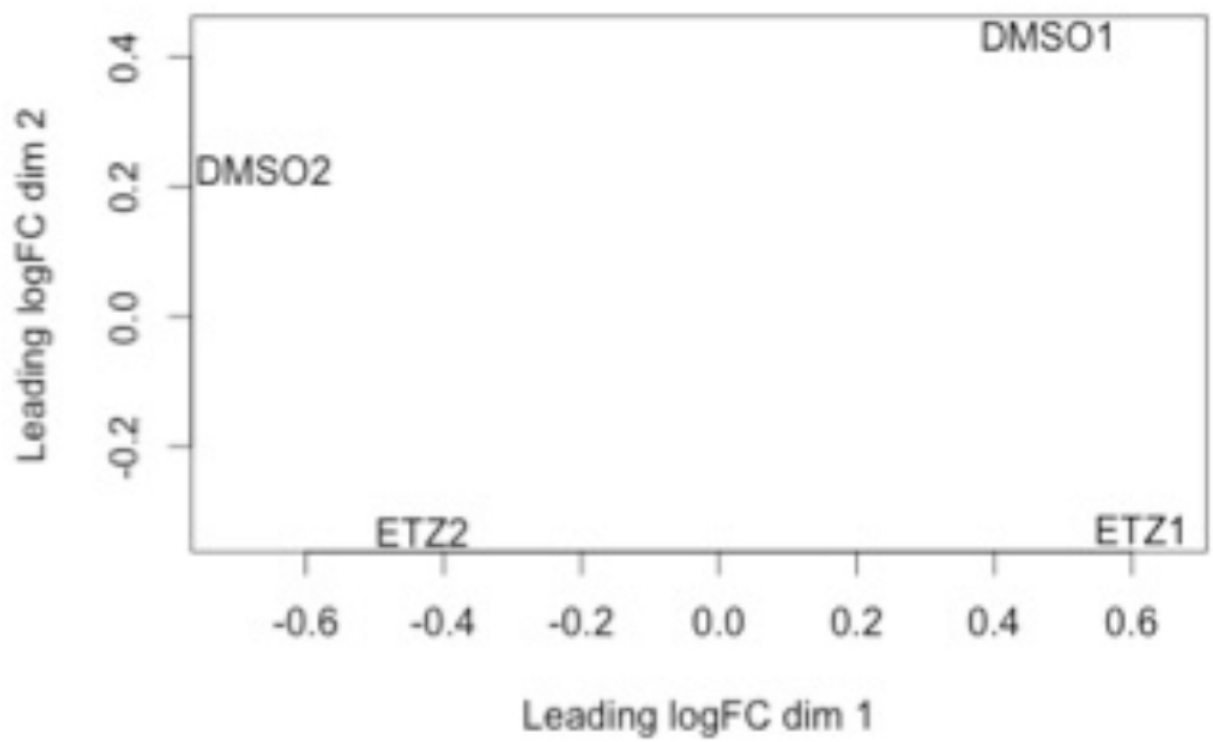
To quote a previously published paper in Nature Reviews Genetics, "Batch effects are sub-groups of measurements that have qualitatively different behaviour across conditions and are unrelated to the biological or scientific variables in a study. For example, batch effects may occur if a subset of experiments was run on Monday and another set on Tuesday, if two technicians were responsible for different subsets of the experiments or if two different lots of reagents, chips or instruments were used."

Thus, it is paramount that one address batch effects within their data before drawing biological conclusions from a specific RNA-seq experiment. To illustrate what a batch effect may look like within the data, we will utilize several different plots.

This first plot comes from the Nature Reviews Genetics paper where they examine Affymetrix data from a published bladder cancer study. You can quickly see that panels C and D from Figure 1 show that samples from batch 1 (blue) cluster together based on gene expression and samples from batch 2 (orange) cluster together.



Within RNA-seq data, using SPARTA and the MDS plot generated by edgeR, another example of batch effects within a study comparing *Mycobacterium tuberculosis* treated with a compound, we can clearly see that the mock-treated samples (DMSO) and compound-treated samples (ETZ) separate based on batch (A vs B) instead of by treatment. Ideally, we would have the samples group together based on treatment as opposed to batch.



If a potential batch effect is detected in the data set, SPARTA will output a message into the terminal that says:

```
IMPORTANT! YOU MAY HAVE A BATCH EFFECT! PLEASE LOOK AT THE MDS PLOT!
```

If this occurs, have a look at the MDS plot in the RNAseq_Data folder -> date of current run -> DEanalysis folder -> MDSplot.png

From here, you will want to adjust your model to account for the batch effect. Within edgeR, this can be accomplished through an additive linear model. The documentation for edgeR contains a tutorial on how to deal with batch effects that can be found here.

Future implementations of SPARTA will include the ability to adjust for batch effects.

Altering Workflow Execution Options

SPARTA is capable of allowing the user to alter the parameters associated with each analysis step to be tailored to specific use cases. Below are the different parameters that can be altered and their usage.

Options:

```
-h, --help            show this help message and exit
--cleanup=CLEANUP     Clean up the intermediate files to save space. Default
                      action is to retain the intermediate files. Usage:
                      --cleanup=True
--verbose             Display more output for each step of the analysis.
--noninteractive      Non-interactive mode. This is for running SPARTA
                      without any user input. Assumes data is on the
                      desktop. If this option is specified, you must fill
                      out the configuration file (ConfigFile.txt) with the
                      appropriate experimental conditions in the SPARTA
                      folder.

Trimmomatic options:
  The order the options will be run are: ILLUMINACLIP, LEADING,
  TRAILING, SLIDINGWINDOW, MINLEN

  --clip=ILLUMINACLIP
                      ILLUMINACLIP options. MiSeq & HiSeq usually
                      TruSeq3.fa; GAII usually TruSeq2.fa. Default is
                      ILLUMINACLIP:TruSeq3-SE.fa:2:30:10. Usage:
                      --clip=<adapterseqs>:<seed mismatches>:<palindrome
                      clip threshold>:<simple clip threshold>
  --lead=LEADING      Set the minimun quality required to keep a base.
                      Default is LEADING=3. Usage: --lead=<quality>
  --trail=TRAILING    Set the minimum quality required to keep a base.
                      Default is TRAILING=3. Usage: --trail=<quality>
  --slidewin=SLIDINGWINDOW
                      SLIDINGWINDOW options. Default is SLIDINGWINDOW:4:15.
                      Usage: --slidewin=<window_size>:<required_quality>

HTSeq options:
  --stranded=STRANDED
                      Stranded options: yes, no, reverse. Default is
                      --stranded=reverse. Usage: --stranded=yes/no/reverse
  --order=ORDER       Order options: name, pos. Usage: --order=name/pos.
  --minqual=MINQUAL   Skip all reads with quality lower than the given
                      value. Default is --minqual=10. Usage:
                      --minqual=<value>
  --idattr=IDATTR     Feature ID from the GTF file to identify counts in the
                      output table Default is --idattr=gene_id. Usage:
```

```
                        --idattr=<id attribute>
  --mode=MODE           Mode to handle reads overlapping more than one
                        feature. Default is --mode=union. Usage: --mode=union
                        /intersection-strict/intersection-nonempty
```

1.1.6 License

1.1.7 Release notes

Version 1.0

1.1.8 Citation

Insert citation here

1.1.9 Acknowledgements

We would like to thank the members of the Abramovitch Lab for helpful discussions and critical assessment/bug identification within the workflow. We would also like to thank the developers and contributors of Python, Trimmomatic, FastQC, Bowtie, HTSeq, and edgeR; without these individuals, SPARTA would not be possible. Finally, we would like to thank you, the user, for utilizing the workflow and making it better.

1.1.10 Functionality wishlist

1. Add paired-end support for SPARTA
2. Add more modular approach to implementing different tools (perhaps through option specification?)
3. Include the ability to deal with batch effects in an efficient manner, requiring minimal user input

- **Contribute:** If you would like to contribute to the project, the source code for each platform can be found in the GitHub repository.
- **Bugs: If you found a bug, please have a look at the issues page and add a description (please be explicit and include error**
  - Mac OS X issues
  - Windows issues
  - Linux issues
- Frequently Asked Questions
- License
- Release notes
- Citation and Acknowledgements
- Functionality wishlist