RNA-Seq Differential Expression Analysis For Non-programmers

Jason W Bacon

RNA-Seq Differential Expression Analysis for Non-programmers

Jason W. Bacon

Contents

List of Figures

List of Tables

First edition April 2025

ISBN-10: 0-9670596-1-5 (ebook)

Published by Acadix Consulting, LLC

https://acadix.biz/

0.1 Acknowledgments

Cover image by Gerd Altmann: https://pixabay.com/illustrations/ai-generated-dna-analysis-research-8789243/

Thanks to Dr. Paul L. Auer and Dr. Ava Udvadia for their thoughtful guidance through my first RNA-Seq differential expression analysis, and to Dr. Andrea Rau for providing a detailed set of example analysis scripts.

0.2 About the author

I'm a quintessential nerd, with BS and MS degrees in computer science, as well as extensive schooling in biology at both undergraduate and graduate levels. I have spent nearly 20 years professionally supporting computational scientific research, including fMRI brain mapping at the Medical College of Wisconsin, and designing, building, and supporting HPC clusters for science and engineering research at the University of Wisconsin -- Milwaukee, an R1 research university. I have more than a decade of university teaching experience in a wide range of computer science topics. While leading HPC support at UWM, I developed popular workshops and a graduate course on research computing, parallel computing, research programming, and parallel programming, for which I wrote The Research Computing User's Guide.

I have contributed directly to biology research on eukaryotic gene regulation and human mosaic chromosomal alterations (mCAs). I am co-author of Mosaic chromosomal alterations in blood across ancestries using whole-genome sequencing, as well as other papers based on my foundational work, which is described at https://github.com/auerlab/TOPMed-mCA.

I have authored several high-performance bioinformatics tools, including ad2vcf (augment VCF files with allelic depth information extracted from SAM/BAM/CRAM files), basic-stats (fast, memory-efficient statistical analyses from the Unix command-line or shell scripts), biolibc (fast, memory-efficient C library for common bioinformatics algorithms and file formats), biolibc-tools (numerous simple tools leveraging biolibc), FASDA (Fast and Simple Differential Analysis without programming), fastq-trim (lightening fast read trimmer), microsynteny-tools (interactive gene neighborhood comparison), peak-classifier (classify ChIP/ATAC-Seq peaks without programming), vcf-split (efficiently split a multi-sample VCF file into thousands of single-sample files in one pass), and vcf2hap (efficient generation of haplotype files use by haplohseq). See https://github.com/auerlab for details.

In addition to bioinformatics tools, I developed Lightweight, Portable Job Scheduler (LPJS), a small and easy-to-use alternative to complicated batch systems like HTCondor, MOAB, Open Grid, Slurm, Torque-PBS, etc., and Simple, Portable Cluster Manager (SPCM), a tool suite for building and managing HPC clusters.

I am also the author of a highly-rated book entitled The C/Unix Programmer's Guide and author of or contributor to numerous other Unix software projects, many of which can be found at https://github.com/outpaddling/.

All that said, I have a non-nerdy side as well, as a high school and college athlete, avid cyclist, cross-country skier, and sea kayaker, scuba diver, private pilot, and long-time student of Kung Fu and Taiji. The common thread running through all of my work and play: It's fun! Most of what I do for its own sake, because I enjoy the experience. In some cases there's a bonus of getting paid to do it. My advice to every young reader: Life should be fun. Work should be fun. Live in the moment. Don't take yourself too seriously. Most of the problems in our world are caused by those who do. When you make mistakes, laugh at yourself and fix them. Do things that you enjoy and at the same time make the world a little bit better. This will lead to a more rewarding career than chasing money, power, or recognition. You'll also likely accomplish more by savoring the experience than by obsessing about recognition or status.

0.3 Goals for the reader

Before undertaking something as challenging as an RNA-Seq analysis, it's important to set realistic expectations. The complexity of gene expression is beyond human comprehension, consisting of thousands of chemical reactions interacting with each other. The results of most biology experiments are uncertain at best. Most of the software tools used in most bioinformatics analyses are far from perfect. Every RNA-Seq analysis will present some unique challenges of its own.

Due to cost restraints, we are often working with only 3 biological samples, which would be considered absurd for most statistical analyses, so confidence in the computational results is generally very low. Unlike TV crime shows, where the boss simply says "enhance image" and three seconds later identifies the bad guy using an online facial recognition database, real software cannot create accurate information out of nothing. In the real world, it's "garbage in, garbage out".

The Pareto principle, also known as the 80/20 rule, states that 80% of the results often arise from 20% of the causes (e.g. work). You can save yourself a lot of wasted time and pointless stress if you limit your initial goal to the following:

Discover something of value in the data.

Don't try to discover everything the data have to tell you on your first try. Data analysis is not like lab experiments, where you only get one shot at it, because you burn through your budget, your samples are destroyed by the experimental procedure, or your reagents have a limited shelf life. You, and others who follow, can always reanalyze the data with improved methods, from a different perspective, or with newer tools, using what you learned from the first run. Doing so may or may not uncover anything new, but trying won't cost much. This is especially true for RNA-Seq, which does not usually require massive computing resources. By gauging your expectations, you can make your way through this text and complete first RNA-Seq analysis without overdosing on caffeine or foregoing personal hygiene.

This book in no way attempts to cover every nuance of RNA-Seq differential expression analysis. You will not be an expert on differential expression after completing this book. You will merely be competent to complete an analysis on your own and obtain useful results. No one will ever master differential expression analysis: There is always more to learn. Everyone who has done an RNA-Seq analysis probably knows something about it that you and I don't. You will undoubtedly learn a few things not covered in this text as you perform your first analysis. With all that in mind, the content of this text is deliberately limited, so that you can complete your first analysis before your grandchildren finish college, and have some depth of understanding about what you did. Future revisions may contain additional material, but the aim of this book will always be to provide a solid introduction, as getting started in bioinformatics is the hardest part. More advanced analysis skills are easier to obtain on your own via other sources once you have a good footing.

Our goal here is simply to get you on your feet in the bioinformatics game, with a clear understanding of the major stages of an RNA-Seq analysis, and at the same time, help you obtain some useful results on your first try. The example provided here will also show you how to perform an RNA-Seq differential expression analysis as efficiently as possible, so that you minimize wasted time and can complete the analysis on a typical PC or Mac. This will help you avoid the need for costly hardware upgrades, and the complexity and bureaucracy of using an HPC cluster or cloud resources. This efficiency also removes barriers to rerunning the analysis with minor changes in order to improve on the results, or to simply explore and learn.

The field of bioinformatics is rapidly changing and will likely continue to do so indefinitely. Much of the software that was mainstream a few years ago is obsolete today. Sequencing methods are constantly improving and becoming cheaper. For these reasons, this book is only available in electronic format, and updates will be made available as often as possible. Feedback is appreciated, so readers should think of this text as a two-way conversation rather than one-way instruction. Questions and comments can be submitted at https://acadix.biz/contact.php.

0.4 How this book is different

Most texts on scientific computing just try to guide you through the status quo, which often resembles a comically impossible obstacle course from a cleverly written Kung Fu movie. They laboriously explain how to use preexisting tools, many of which are of poor quality and/or extremely difficult to install, learn and use.

In contrast, my goal from the beginning has been to make the entire RNA-Seq differential analysis process, from software installation through comparison of gene expression, as easy as possible. My 20 years supporting computational science has taught me that most tasks are a lot harder than they need to be. So, my goal here is to demonstrate an easier approach for one of those tasks, namely differential expression analysis. This is where my background in software and systems engineering comes in handy, as well as my experience providing I.T. support to a wide variety of scientific researchers over almost 20 years. I have

a rare perspective on how scientists struggle needlessly with computer analyses in general, as well as the software engineering skills to do something about it.

All of the software tools necessary for the analysis pipeline presented here can be installed with a single command, on any popular Unix-compatible system (following some simple prep work on many systems). How this was made possible is explained in the chapters that follow.

Some of the traditional tools used in RNA-Seq differential expression analysis are well-maintained, well-documented, and fairly easy to install and use. Some examples include general bioinformatics tools like samtools, bedtools, and vsearch, and read mappers like bwa and bowtie2. I have written new tools to replace the ones that cause unnecessary difficulty, and have done so in a sustainable way. The tools I have written use programming languages and coding standards to ensure that minimal maintenance will be required in the future. They will continue to be easy to build and install long after my atoms are scattered across the planet. This is a stark contrast to much scientific software, which becomes unusable within a few years after the authors stop maintaining it. Many of these problems are due to the use of trendy programming languages that fail to maintain compatibility with prior versions, or fall out of favor entirely, being replaced by the next "messiah" language.

Just as importantly, I have designed these tools to be *used* in a sustainable and balanced way, aiming to *minimize* what you need to learn about Unix, but not *eliminate* it. Some people, with the best of intentions, have made attempts to create of graphical user interfaces (GUIs) that anyone can use to do bioinformatics with zero training. Invariably, though, they are poorly implemented and maintained, often providing outdated software, and incomplete, cumbersome interfaces. What you can do with a GUI is limited to the features that the developers have time to create and maintain. Just building a full-featured GUI is a massive programming effort, but what most scientists are unaware of is that maintaining it over the coming years is often harder, due to the ever-changing programming languages and other software on which it depends. Neither the funding nor the talent pool for all this development and maintenance work exist.

Creating Unix command-line tools, in contrast, requires minimal programming effort without imposing any artificial limits on the user. They allow you full access to all of the functionality of all of the programs you use. This is why many of the most evolved and stable tools (samtools, bowtie, etc.) out there are command-line based.

I could go on for weeks about all the reasons to learn the Unix command-line, but let's focus on the only reason we really need:

There will always be things that you can only do from the command line, for the reasons described above.

If you try to avoid learning the command-line, you will at best delay the inevitable until it becomes urgent, which will only create a stressful situation. Meanwhile, you will waste time learning other esoteric and ephemeral interfaces that will be of no use to you a few years from now. The Unix command-line is here to stay, largely unchanging, and extremely empowering as a standard used by hundreds of thousands of programs. Once you understand how the Unix command line works, it will be easy to learn and use countless programs that you might need in the future. Invest a little time to learn command-line basics now, and you will be prepared to tackle future analyses quickly when you have an impending grant deadline.

0.5 Prerequisites and corequisites

The only requirements for following this text and completing the RNA-Seq differential analysis like the one presented here, are a solid background in genetics, and basic knowledge of the Unix shell environment (command line), and a typical PC or Mac computer. Unix shell scripting, which is simply a matter recording sequences of Unix commands in a file for easy repetition, will also be useful.

Note

Throughout this text, the string "shell-prompt: " is used to indicate that the following text is a Unix command. On your Unix system, "shell-prompt: " will be replaced by a different prompt, but in either case, this is the Unix shell (command interpreter) telling you that it is waiting for you to enter another command.

In some cases, there will be some text following a '#' character above or next to the command. These are comments to explain the command, not part of the command, so you don't type this stuff in.

A solid grade in a college genetics course should be sufficient to provide the necessary biology background. This should be completed *before* trying to tackle an RNA-Seq analysis. Without this, you will not be able to understand the concepts discussed here, such as transcription, RNA splicing, etc.

The Unix background is more of a corequisite: You should be comfortable with running commands from the Unix command-line before diving into RNA-Seq, but you don't need to memorize a lot of commands beforehand. The exact commands you need to complete the differential analysis are provided here. You should understand what a shell script is (basically a file containing Unix commands). You can then grow your knowledge gradually over the rest of your career. Unix basics can be quickly gained by studying Part I of The Research Computing User's Guide. There are many other sources of Unix knowledge available for free, but most of them are sketchy and there are no others that I'm aware of that are thorough and geared toward scientific researchers.

For those of you who have no idea what Unix is: In a nutshell, it's every operating system you're likely to use, except Microsoft Windows. Unix was originally a trademark for a specific operating system developed at AT&T Bell Labs, but the term now represents a set of standards to which most operating systems (but not MS Windows) conform. The primary Unix standard is *POSIX*, an acronym for "Portable Operating System Standards based on Unix".

The Unix family includes many commercial operating systems including Apple's macOS, as well as many free systems, such as BSD (Berkeley Systems Distribution) derivatives like Dragonfly BSD, FreeBSD, NetBSD, OpenBSD, etc. Linux-based systems such as Debian, RHEL, and Ubuntu, are also Unix compatible. Apple's macOS since version 10.0 (the first OS X), is largely based on FreeBSD, and complies with most Unix standards. What these standard do, primarily, is ensure that a program written on one Unix-compatible system will work on all of them with little or no modification. Hence, a program written on Linux will work on BSD and macOS. A program written on BSD will work on macOS and Linux. A program written on macOS will work on BSD and Linux (provided only standard Unix features were used, and not Apple's proprietary extensions). The standards also ensure that the user interface is mostly the same, so most Unix commands will work the same on BSD, Linux, macOS, etc.

So, if you have a Macintosh computer, you're ready to get started. If you don't have a Mac or access to some other Unix system, it won't cost you anything to get one. You can run a free Unix system on any modern PC, directly on the hardware, or using a free virtual machine monitor such as VirtualBox on your Windows PC. Chapter 3 provides basic instructions for installing a Unix system and the RNA-Seq software. The Research Computing User's Guide contains more extensive information on virtual machines.

0.6 Practice Problem Instructions

- Practice problems are designed to help you think about and verbalize the topic, starting from basic concepts and progressing through real problem solving.
- Use the latest version of this document.
- Read one section of this document and corresponding materials if applicable.
- Try to answer the questions from that section. If you do not remember the answer, review the section to find it.
- Do the practice problems *on your own*. Do not discuss them with other students. If you want to help each other, discuss *concepts* and illustrate with different examples if necessary. Coming up with the correct answer on your own is the only way to be sure you understand the material. If you do the practice problems on your own, you will succeed in the subject. If you don't, you won't.

  If you're still not clear after doing the practice problems, wait a while and do them again. This is how athletes perfect their game. The same strategy works for any skill.
- Write the answer in your own words. Do not copy and paste. Verbalizing answers in your own words helps your memory and understanding. Copying does not, and it demonstrates a lack of interest in learning.

  Answer questions completely, but *in as few words as possible*. Remove all words that don't add value to the explanation. Brevity and clarity are the most important aspects of good communication. Unnecessarily lengthy answers are often an attempt to obscure a lack of understanding and may lead to reduced grades. "If you can't explain it simply, you don't understand it well enough." -- Albert Einstein
- Check the answer key to make sure your answer is correct and complete.

  DO NOT LOOK AT THE ANSWER KEY BEFORE ANSWERING QUESTIONS TO THE BEST OF YOUR ABILITY. In doing so, you only cheat yourself out of an opportunity to learn and prepare for the quizzes and exams.
- ALWAYS explain your answer. No exceptions. E.g., justify all yes/no or other short answers, show your work or indicate by other means how you derived your answer for any question that involves a process, no matter how trivial it may seem, draw a diagram to illustrate if necessary. This will improve your understanding and ensure full credit for the homework.

- Verify your own results by testing all code written, and double checking short answers and computations. In the working world, no one will check your work for you. It will be entirely up to you to ensure that it is done right the first time.
- Start as early as possible to get your mind chewing on the questions, and do a little at a time. Using this approach, many answers will come to you seemingly without effort, while you're showering, walking the dog, etc.
- For programming questions, adhere to all coding standards as defined in the text, e.g. descriptive variable names, consistent indentation, etc.

Chapter 1

Bioinformatics analysis background

1.1 The status quo

Bioinformatics, the computational analysis of DNA/RNA sequence data, has become central to numerous areas of biology research in recent years. Unfortunately, the computer skills required for many analyses using traditional software are far beyond those of a typical lab or field biologist. Simply installing the necessary software often requires advanced I.T. skills, and simply using many of the traditional analysis tools requires significant computer programming skills. This is an unnecessary waste of your valuable time, and can be a major source of stress.

RNA-Seq differential expression analysis is inherently challenging. However, many existing software tools make it vastly more difficult than it needs to be, in part because they require you to become a computer programmer just to use them. If you plan to make a career of bioinformatics, then solid programming skills will be essential. But many common analyses, including RNA-Seq differential expression, can, with the proper tools, be performed with nothing more than basic Unix skills. If you don't know how to program, this book will allow you to complete an RNA-Seq analysis anyway. If you do know how to program, this book will make the analysis faster and easier, leaving more of your valuable time for other tasks.

The quality of bioinformatics software ranges from outstanding to so bad that developers are tempted to change their names for the sake of plausible deniability. Many analysis software projects are created in order to complete a research project and publish a paper, often by a graduate student. The software project is often abandoned after the paper is published, or the student graduates and discovers that someone will actually give them money to do something else. In general, software that is no longer supported is known as *abandonware*. Software that was developed in order to publish a paper or finish a thesis is a special category of abandonware known as *paperware*. Without ongoing maintenance, most software quickly becomes non-functional due to inevitable changes to other software on which it depends, the programming languages used, and the operating systems on which it runs. Use of unsupported software can make it difficult or impossible to reproduce analysis results as little as a few years later. Most scientists don't realize that writing software is like adopting a puppy: It requires a long-term commitment to keep it healthy and well-behaved, and it will leave many surprises for you the rug, especially in the early days.

Finding a collaborator or hiring a bioinformatician is often not an option either, since people with the necessary skill set are in short supply. They are either too busy, or, due to the basic law of supply and demand, cost more than the average researcher can afford. Until biologists can perform more of their own bioinformatics analyses, a great deal of research will remain stuck in the mud due to this software hurdle.

A great deal of time in the bioinformatics field is wasted on duplicated effort due to a shortage of well-developed application software for performing common tasks. Many bioinformaticians code their own minimalist, single-use solutions to common problems, never developing the code into a generalized application that others could use. As a result, many others waste time coding similar solutions for their own purposes. In order for the bioinformatics community to evolve toward greater productivity via less duplicated effort, this needs to change. We need more well-organized projects like **samtools**, **bedtools**, **vsearch**, **bwa**, and **bowtie2**, just to name of few of the positive examples already out there. Focusing on long-term solutions rather than short-term gains would benefit everyone in the field and move a great deal of research forward faster. There will likely always be a shortage of bioinformaticians. Allowing bioinformaticians to focus on novel problems rather than duplicating each others' quick-fixes will be a crucial step toward making better use of our valuable time. The problem is, investment in better bioinformatics development, which would would be alleviate the shortage of manpower in the long-term, is discouraged by the need to meet short-term deadlines. We have all taken the path of least resistance for the sake of impending deadlines, knowing full-well that a

more thorough approach would be better for the long-term. It's a conundrum, but one we must solve in order to move the field forward.

1.1.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. Why has RNA-Seq differential expression analysis traditionally been difficult for the average biologist?
2. What is "abandonware"?
3. What is "paperware"?
4. Why don't biologists just find a collaborator with bioinformatics experience to help with computer analyses?
5. What is needed in the bioinformatics community to make analyses easier?

1.2 A much easier approach

This text presents an example solution for one of the most common analysis types in bioinformatics: RNA-Seq differential expression analysis. We show how an RNA-Seq analysis can be brought within reach of typical lab biologists with little or no computer programming background, by making software installation trivial, software use much easier, and by writing the software in a way that requires minimal ongoing maintenance. The tools and methods presented here reduce the computer skills required for an RNA-Seq DE analysis to a minimum, which can be easily learned in one semester. Rather than having to learn Unix, R programming, python and/or perl programming, bioconductor and R-cran, and the esoteric I.T. skills needed to debug failed software builds, you need only know a little bit of Unix in order to complete the analysis presented here.

Note If you plan to do bioinformatics for a living, you will need to learn R scripting, and other languages like C, Perl, and Python as well. Our purpose here, though, is to allow the average biologist to do their own differential expression analysis in a reasonable amount of time, on a typical PC or Mac. No programming, beyond some simple Unix shell scripting, is inherently necessary for this purpose.

If all goes well, a typical RNA-Seq differential analysis should take no more than a few days on a modern PC, from downloading the raw sequence data, to having fold-changes and associated P-values for every known gene or transcript. RNA-Seq differential expression analysis does not usually require massive computational resources (though some bioinformaticians intent on selling you their services might tell you otherwise). In fact, most bioinformatics can be done on an ordinary Mac or PC, if using efficient software tools. One notable exception is de novo genome assembly, which often takes several days using hundreds of processors on an HPC cluster (anywhere from several to thousands of servers working in parallel). A de novo assembly of a large genome is akin to assembling a one billion piece jigsaw puzzle with all the pieces upside-down.

The solution presented here renders software installation trivial by leveraging *package managers*, which automate the installation, removal, and upgrade of a software package, and all others on which it depends. This completely eliminates the struggles with software installation that I have seen so many researchers battle in agony, often giving up in the end. Most researchers don't have easy access to someone like me to solve these problems for them. So, I dream of a world researchers can easily install any program they need in a matter of minutes, and then can focus their incredibly valuable time on scientific discovery, rather than completely avoidable I.T. problems. This is not a pipe dream. It is absolutely possible, using a fraction of the vast resources and talent that are currently being wasted on primitive, disorganized software installation. All we have to do is redirect those resources to a more intelligent approach.

Our use of package managers here is meant to provide an example of how easy software deployment can (and should) be, so that more scientists will realize that the don't have to tolerate being blockaded by the status quo, and start doing things to improve on it. A package manager is a prime example of a great investment: We put some added effort into creating a package ahead of time, and get a many-fold payback by spending seconds, rather than hours, on every future installation. The best approach, if possible, is to use a single, unified package manager for all of your open source software needs, rather than multiple different package managers. This isn't always feasible, but it is for our RNA-Seq analysis. Doing so brings the following benefits:

- It makes software installation trivial, even for the most complex software packages.
- It eliminates *context switching*, where we switch between various software installation methods, causing confusion and frequent mistakes.
- It eliminates conflicts between software installed by different methods, where a program might inadvertently use the wrong version of another program or library installed by different means.
- It greatly simplifies every user's shell environment, since all of their software is installed in one place. They don't need different directories in their PATH in order to run different programs.

Note that I personally created some of the FreeBSD ports and dreckly packages required for this analysis. This might make some people wonder if the system presented here has a "bus factor" of one. (The "bus factor" is the number of people who need to get hit by a bus in order to doom a project.) However, continued support for these packages does not depend on me alone. At the time of this writing, there are 1,669 people maintaining 33,268 FreeBSD ports, 21 of whom contribute the 256 ports in the biology category. The biology ports also utilize hundreds, if not thousands, of ports in other categories indirectly. You might also directly use software from the math category (such as the R statistical language), which has 108 maintainers and 1,300 ports, and science, which has 33 maintainers and 527 ports. Not to mention all the other ports you might use for non-scientific tasks, such as LibreOffice, Firefox, Thunderbird, etc. The situation is similar for dreckly, pkgsrc, and other popular package managers such as Debian packages, MacPorts, etc. *You* can learn to create and maintain packages as well. Many of them require only minimal computer skills, and every popular package manager has a vibrant community full of people eager to help you learn and improve. The effort required to create a package will likely pay you back at least ten-fold on future installations of the software, and help your collaborators around the world expedite research as well. Using package managers is by far the most sustainable approach to managing open source software.

We also developed some new software tools to replace those that require programming skills just to use. Our tools are much easier to install, easier to use, and more efficient than the traditional tools they replace. They are also designed to require minimal maintenance, so they will continue to function many years into the future, allowing research results to be easily reproduced. This efficiency and forward compatibility comes from coding in C and POSIX Bourne shell, rather than one of the trendy scripting languages, which are typically about 100 times slower than C, and often break compatibility when the next version of the language is released. Both C and Bourne shell are extremely stable (not changing in ways that will cause programs or scripts to stop functioning under a future version of the language), and extremely portable (able to run on different operating systems and hardware). C is the world's fastest and most memory-efficient portable programming language (see https://github.com/-outpaddling/Lang-speed/tree/master/Results). Bourne shell is a standard component of every Unix-compatible system.

Use of these tools and installation methods enables more biologists to perform their own differential expression analyses with minimal delay and distraction, so they can remain focused on the biology research to which they chose to dedicate their lives.

1.2.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

---

1. What are three ways the approach in this text brings RNA-Seq differential expression analysis within reach of the average biologist?
2. What is the minimum computer knowledge required to perform an RNA-Seq differential expression analysis?
3. What computer knowledge is required to become a professional bioinformatician?
4. What kind of computer hardware is necessary for a typical differential expression analysis, and how long will it take to complete?
5. How does the approach used by this book facilitate software installation?
6. Describe four ways the use of a unified package manager helps scientists be more productive.
7. Why do the new tools created for this analysis approach require less maintenance than most?

1.3 How you can help

The easiest and most important step scientists can take to improve computational science is to simply ask for better software, and simpler, more reliable installation methods. If a tool is difficult to install, difficult to use, or doesn't work, *don't just accept the situation*. Report the problem to the developers so that they're aware that there is a problem. Only the most obsessive (or bored) developers will spend time improving their software when they believe people are happy with it. The more users report problems, they more likely they are to be fixed.

Ask developers to add their software to popular package managers such as Debian packages, dreckly, or FreeBSD ports. This will help both you and your colleagues, and will help the developers by eliminating many help requests for manual software installations.

Some developers will respond immediately, while others may not respond at all. Don't let a few unresponsive developers discourage you from reporting problems in the future, though. View it as a long-term, general strategy for making your career more productive, not a short-term fix for this one particular problem. Most of your problem reports will have a positive impact. I know this from decades of experience working with numerous developers.

1.3.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

---

1. What is the easiest way for users of scientific software to help improve software quality for everyone?

1.4 What is a pipeline?

An *analysis pipeline*, in the broadest terms, is any procedure involving a sequence of processing stages to obtain results from raw data. Output from one stage is input to the next. A pipeline can be a manual process, in which the researcher runs each stage individually by hand, or automated, in which case all the stages are scripted to run in sequence without user interaction. It could also be anything in between.

Automated pipelines are appealing, but not usually ideal. With automated pipelines, the researcher is a bystander dealing with a *black box*, having little to no understanding of the analysis details. Automated pipelines can run for hours, days, or even weeks, and if the early results are not usable, the time spent on later stages is wasted, along with the extensive computing resources that might have been better used for other tasks.

A manual pipeline keeps the researcher fully engaged in the analysis, following the *know your data* principal. With a manual pipeline, the researcher runs one processing stage, and then carefully examines the results to determine whether the quality is sufficient to justify moving on to the next stage. In this text, we adhere to the "know your data" principle, and will take you through a manual RNA-Seq differential analysis pipeline, performing some basic quality checks after each stage.

1.4.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

---

1. What is a pipeline?
2. What is the main problem with automated pipelines?

1.5 FASTQ files

RNA-Seq is a biological procedure for extracting RNA from biological samples, fragmenting it, and using a sequencing machine to read full or partial sequences from as many fragments as possible. Like many lab protocols, the process depends on a bit of luck and the results are never perfect. The exact timing of RNA extraction will affect RNA abundance. Not every fragment will be read by the sequencer, and some will be read more than once.

The sequencing machine is usually run by a dedicated sequencing professional, and outputs *FASTQ* files. A FASTQ file is an ordinary text file that contains a series of *reads*, each of which represents *part* of the sequence from a DNA or RNA fragment that was in the solution loaded into the sequencer. Each read contains four *fields*:

- A line starting with the '@' character followed by a unique sequence identifier and optional text describing the read. This line has no standard format, but typically contains information about the sample and the sequencing run.
- One or more lines of sequence data.
- A separator line beginning with '+', optionally followed by the same identifier and description in the first field. As sequence data cannot contain a '+' character, this marks the end of the sequence.
- One more more lines containing quality scores, totaling the exact same length as the sequence. The end of the scores data is indicated by the '@' marking the beginning of the next read, or by the end of the file. The quality scoring system is described below.

Note FASTQ format should not be confused with *FASTA* format, which is very similar, but contains only base sequences, not quality scores.

Figure 1.1 shows two sequences of fifty bases each in FASTQ format.

```
@generand0
TGCGAACACAGGAGCATTACAGATA
TAATACGGGCTGGTAGGGGAAGTTG
+generand0
===============<?????????
????::::==========;;3G0C
@generand1
CAGTGCCGCTCAGGTGACTAAAATG
GAGCCAGCTCTAGCCCGCTCAGCCC
+generand1
@@@@@@@@@@@@@@@@@@@@@@@@@FG
GGGGGGGGGGGGGGGFFFFFF25<:
```

Figure 1.1: FASTQ file contents

Caution It's a common assumption that FASTQ files contains entries of four *lines*, with all the bases on the second line and all the quality scores on the fourth line. This is not true. It is often the case, but the FASTQ standard does not require all the bases or all the quality scores to be on one line, so we must not assume this format when writing software to read FASTQ files. There are many existing FASTQ files that have sequences and quality scores spread over multiple lines. We write software once, and use it potentially millions of times, so when should we deal with the possibility of multiline sequences?

The quality scores represent the sequencing machine's estimate of the probability that the base was read correctly. Scores are encoded in *PHRED* format, which represent each score as a printable character. The quality corresponds to the character's position in the standard ISO character set. For example, 'A' is at position 65, 'B' at 66, and so on. Higher scores mean that the sequencing machine estimated a higher probability that the base was read correctly.

Note How trustworthy the machine's quality estimates are is a matter of much debate. There are many things that could go wrong while determining the score. However, we don't have much else to go on, so we generally have to trust the scores we are given, at least in the early stages of the analysis. A few incorrect bases usually don't impact the analysis much, and subsequent analysis stages may reveal problems that the sequencer missed.

PHRED is a logarithmic scale ranging from 0 (the read is almost certainly wrong for this base) to 40 (it is astronomically unlikely that a read error occurred). The probability of a read error (assuming we trust the sequencing machine's estimate implicitly) is

$P(error) = 10^{-phred/10}$.

Typically, a score of 20 or higher is considered acceptable, and 28 or higher is considered very good.

$P(error) = 10^{-20/10} = 10^{-2} = 0.01$, indicating a 1 in 100 (1%) chance that the base was read incorrectly.

$P(error) = 10^{-25/10} = 10^{-2.5} = .00316227766016837933$, indicating about a 3 in 1000 (0.3%) chance that the base was read incorrectly.

$P(error) = 10^{-28/10} = 10^{-3} = 0.0015848931924611348$, indicating a 1.6 in 1000 (0.16%) chance that the base was read incorrectly.

Note If you're wondering why the exponent is divided by 10, no it is not just to daze and confuse us. It is to increase the resolution of the PHRED scoring system, which uses a rather large log base of 10, and where the scores must be integers in order to map to an ISO character set. If we did not divide by 10, then the integer scoring system could only represent probabilities of 0.1, 0.01, 0.001, etc., which would be too coarse. Dividing by 10 means we can represent nine more values between 0.1 and 0.01.

Inverting the function $P(error) = 10^{-phred/10}$:

1. $\log_{10}(P(error)) = -phred/10$
2. $10 * \log_{10}(P(error)) = -phred$
3. $phred = -\log_{10}(P(error)) * 10$

Since characters 0 through 32 in the standard character ISO sets are invisible (e.g. 0 = nul, 9 = tab, 10 = newline, 13 = carriage return, 32 = space), PHRED adds a "base" (usually 33) before converting the integer value to a character for storage in the FASTQ file. Using visible characters doesn't matter at all to a computer, but it allows us humans to "see" the scores when we look at the FASTQ text file. Conveniently, this makes the worst possible score of 0 translate to the character '!', which seems intuitive. Beyond that, a PHRED score of 1 is stored as character 34 ('"'), 20 as character 53 ('5'), and 30 as character 63 ('?'). Not exactly intuitive, is it? But, at least it allows us to figure out the quality of a read, with the aid of a character set table and a little math. You'll also learn to recognize good and bad scores at a glance with enough practice viewing FASTQ files.

The quality scores in Figure 1.1 are generally high, e.g. '=' is character 61, indicating a score of 61 - 33 = 28 (P(error) = .00158), and '@' is character 64, indicating a score of 31 (P(error) = .00079).

We often see lower scores at beginning and/or end of each read, due to the chemistry of the sequencing process being out of equilibrium at the extremes. If the drop in quality is significant, we can simply discard these bases during the read trimming process. The Illumina documentation has some good explanations and cartoon videos of the sequencing process.

1.5.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. How many lines represent each read in a FASTQ file? Explain.
2. Describe the four fields of a read in a FASTQ file.

3. What is the *numeric* PHRED score and the probability of a read error for the fifth base in the following read, assuming PHRED-33 scoring. Reminder: As for all practice questions, show your work.

```
@generand1
CAGTGCCGCTCAGGTGACTAAAATG
GAGCCAGCTCTAGCCCGCTCAGCCC
+generand1
@@@@@@@@@@@@@@@@@@@@@@@@@FG
GGGGGGGGGGGGGGGFFFFFF25<:
```

4. Are quality scores typically consistent for a given read? Why or why not?

5. What is generally considered a high quality score? An acceptable score?

1.6 The sequencing procedure

First, we collect RNA samples from tissue or single cells. There are well-developed protocols for collecting and purifying DNA/RNA, though it may require some practice before you can produce quality results. In the context of RNA-Seq analyses, the collection of resulting DNA/RNA samples is called a *library*.

Sequencing machines and supplies cost money, and require extensive training, so the sequencing step is typically outsourced to specialists rather than done locally. We usually send our samples (library) to a sequencing center, where sequencing specialists prepare the samples by adding *adapters*, artificial sequences appended to each end of each fragment, which we call the *insert*. The adapters contain fixed start and end sequences required by the sequencing machine, and possibly also *tags* (sequences specific to each sample), which will allow us to determine the sample from which each read originated. (The samples may all be mixed in the sequencer to save time and money, so we need a way to distinguish them in the FASTQ files.) Figure 1.2 shows a simplified version of what typical DNA sequence might look like after adding adapters.

```
          --- 5'-adapter ---
          Fixed        Tag   Original fragment (insert)          3'-adapter
Sample 1: AGATCGGAAGAG ATCTA ATGCATGCATACACCTAGTCGTAGTAGCTAGCTA AGATCGGAAGAG
Sample 1: AGATCGGAAGAG ATCTA TAGCTAGCTAGCTAGCTAGCGATTCAGTA      AGATCGGAAGAG
...
Sample 2: AGATCGGAAGAG TAGCT TGATCGTCGATGCTAGCTAGTAAGCTAGCTGT   AGATCGGAAGAG
...
```

Figure 1.2: Sequencer DNA structure

The sequencing specialist then loads the samples into the sequencer, and when the run completes, sends us FASTQ files containing partial sequences of many (but not all) of the fragments.

DNA/RNA samples are fragmented by endonucleases before being sequenced. The size of DNA/RNA fragments varies greatly, as the recognition sites for endonucleases are scattered more or less randomly throughout the genome or transcripts. Fragment length typically averages a few hundred bases. The base sequences in DNA are not truly random (whatever "truly" random actually means), of course, as natural selection favors sequences that actually work. However, there is no obvious pattern in the location of recognition sites, so fragment length can be considered essentially random.

The sequencers used for RNA-Seq analyses usually produce relatively short, fixed-length reads, often between 50 and or 150 bases long. If the fragment/insert + adapters is shorter than the read length (Figure 1.3), the sequence is read entirely and the read may contain trailing random "garbage" sequences at the end, depending on the sequencer used. The random sequences will follow the 3' adapter, so they are easily recognized and removed.

```
# Fragment + adapters shorter than the read length
Fragment: TAGCTAGCTAGTCGATCTAGCTACTAG
Read:     TAGCTAGCTAGTCGATCTAGCTACTAG 3'-adapter GATCGTAGCTAGCTGATCGA
                                                 ^^^^^^^^^^^^^^^^^^^
                                                        Garbage
```

Figure 1.3: Fragment shorter than read length

Sequences longer than the read length will be read only partially (Figure 1.4), so portions of many fragments in our library will not be represented at all in the FASTQ files.

```
# Fragment longer than the read length
Fragment: ATCGTAGCTAGCTTACGTAGCTGACTAGCTAGTCATCTAGCTAGCTAGCTAGCTAGCTGATCG
                                                          ^^^^^^^^^^^^^^^^^^^
                                                                 Unread
Read:     ATCGTAGCTAGCTTACGTAGCTGACTAGCTAGTCATCTAGCTAGCTA 3'-adapter
```

Figure 1.4: Sequence longer than read length

This means that the data in the FASTQ files do not accurately represent the sequences in the library. Large portions of longer fragments are essentially lost during sequencing. However, there will be many copies of each transcript from an active gene in the library, so it is highly likely that some copies of each portion of each transcript will be sequenced. This is enough to give us a rough idea about how active a gene was shortly before sampling occurred.

1.6.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. Describe the three major steps in the RNA-Seq lab protocol.
2. What is an adapter?
3. What is a tag?
4. What is an insert?
5. Do sequencers typically sequence entire mRNAs?
6. What is a typical read length for RNA-Seq?
7. Are all the bases in our fragments represented in typical FASTQ files? Why or why not?

1.7 Single vs paired-end sequencing

In *single-ended* (SE) sequencing, fragments are sequenced starting only from the 5' end. If the fragment, including adapters, is longer than the read length, part of the 3' end will be lost during sequencing.

Paired-end (PE) sequencing is a method used to obtain more information about fragments longer than the read length. In paired-end sequencing, both ends of each fragment are sequenced, rather than just the 5' end, reads. Reading the 3' end involves some biochemical acrobatics, but it is worthwhile for the added information we get from it. The forward and reverse reads from the same fragment are known as *mates* and the two together are called a *mated pair*. The 5' read is called the *forward read* and the 3' read is called the *reverse read*. Relative to the actual fragment that was sequenced, we usually end up either with a gap between

the mates (if the fragment length with adapters is more than twice the read length, Figure 1.5) or an overlap of the two mates (if the fragment length with adapters is shorter than twice the read length, Figure 1.6).

```
Fragment: GATCGTAGCTGTCGATCAGTCGATCAGCTAGCTAGTCGATGCTAGCATCTACTAGTCGTACTAGC
SE read:  GATCGTAGCTGTC
PE reads: GATCGTAGCTGTC                                        TAGTCGTACTAGC
          ^^^^^^^^^^^^^                                        ^^^^^^^^^^^^^
          Forward read                                         Reverse read
```

Figure 1.5: Gap in paired-end sequencing

```
Fragment: GATCGTAGCTTAGTCGTACTAGC
SE read:  GATCGTAGCTGTC
PE reads: GATCGTAGCTGTC
          ^^^^^^^^^^^^^
          Forward read
                    TAGTCGTACTAGC
                    ^^^^^^^^^^^^^
                    Reverse read
```

Figure 1.6: Overlap in paired-end sequencing

Ending up with two mates that concatenate exactly to the insert size is about as likely as winning the lottery.

In paired-end sequencing, which is now the norm in bioinformatics, *the sequencer produces two separate FASTQ files with the exact same number of reads*. The 5' (forward) reads, such as the GATCGTAGCTGTC above, are in one file, and the 3' (reverse) reads, such as the TAGTCGTACTAGC above, are *at the exact same position* in the other file.

Caution

The position within the FASTQ file is the only way to identify the mate of a given read. Tools that alter paired-end FASTQ files must keep the two files in sync. E.g., if a trimming tool removes an entire read from one file, it must also remove the mate from the other file. Most modern bioinformatics tools support PE reads, but we must be sure to use them correctly to avoid corrupting our data.

When both the forward and reverse reads are properly mapped to a genome or transcriptome, we can determine the *length* of the sequence between them, though we don't know its content. With a little luck, the content can be filled in by other reads that mapped to this location, assuming the *depth of coverage* (number of reads from the same location) is sufficient. If the reads don't all agree on a given base (due to read errors or actual differences in the DNA/RNA), we can use *consensus* to decide what the base should be. I.e., each read essentially gets a vote, and the majority wins. We might be tempted to fill in the gap using sequence from the reference (Figure 1.7), but this would essentially be fudging the data, so we need to be careful. The reference comes from different individuals than our samples did, so there will be differences between the DNA sequences.

```
Fragment: GATCGTAGCTGTCGATCAGTCGATCAGCTAGCTAGTCGATGCTAGCATCTACTAGTCGTACTAGC
SE read:  GATCGTAGCTGTC     |                |
PE read:  GATCGTAGCTGTC     |                |                TAGTCGTACTAGC
          ^^^^^^^^^^^^^     |                |                ^^^^^^^^^^^^^
          Forward read      |                |                Reverse read
Reference:GATCGTAGCTGTCGATCACTCGATCAGCTAGCTAGACGATGCTAGCATCTACTAGTCGTACTAGC
                          ^                ^
                  Differences between fragment and reference
```

Figure 1.7: Problem with using reference to fill gaps

1.7.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. What is paired-end sequencing?
2. Describe two advantages of paired-end sequencing.
3. How are paired-end reads represented in FASTQ files? What complication does this present for analysis?

Chapter 2

RNA-Seq pipeline overview

In this chapter, we present a brief conceptual overview of an RNA-Seq differential expression pipeline, to help you understand the the major goals and analysis steps. Later, in Chapter 4, we go through a real analysis in detail, demonstrating how to perform each analysis stage and verify the results. Before going into such detail, it is important to have a perspective of the big picture. This will help you understand how what we do at each stage prepares the data for the next stage. The repetition of this overview and the more detailed example in Chapter 4 should also help you gain a deeper understanding.

2.1 The pipeline

RNA-Seq experiments estimate (not measure) the *abundance of RNA transcripts* from each gene in a microbial colony, tissue sample, or even a single cell, at the moment of sampling. These estimates are far from accurate in most cases, due to many factors beyond our control, such as RNA degradation rates, and some within our control, such as lab technique. However, accuracy isn't always critical, such as when comparing abundances across two conditions where the actual expression rate changes 10-fold. If, on the other hand, we see a 1.5x change in abundance for a given gene, we are left to wonder whether this represents a real change in expression, an artifact in timing of the sampling, or variability in the purification process.

RNA abundance is an estimate of gene activity, but not necessarily the best one for every purpose. We use it mainly because it is the easiest metric to gather using tools available at the time of this writing. It does not necessarily correlate to transcription rate, for example, because RNA degradation rates vary widely and directly affect RNA abundance at the moment of sampling. It does not necessarily correlate to protein abundance either, since translation rates and protein degradation rates are also highly variable. As with so many biological processes, we cannot directly observe what we want to know, so we measure what we can and make inferences from the available data.

One of the most common uses for RNA-Seq data is the *differential expression analysis*, where the RNA abundance for each gene or transcript is compared across two or more conditions, such as time points during development, wild-type and mutant strains, or normal and stressed states (perhaps due to extreme temperatures or exposure to a toxin).

Caution
Results from an RNA-Seq differential expression analysis are never conclusive. They are only suggestive, helping us determine which apparent changes in gene expression are our best bet for utilizing limited resources on experimental verification.

A typical RNA-Seq pipeline involves several major steps, depicted in Figure 2.1, and summarized in the sections below.

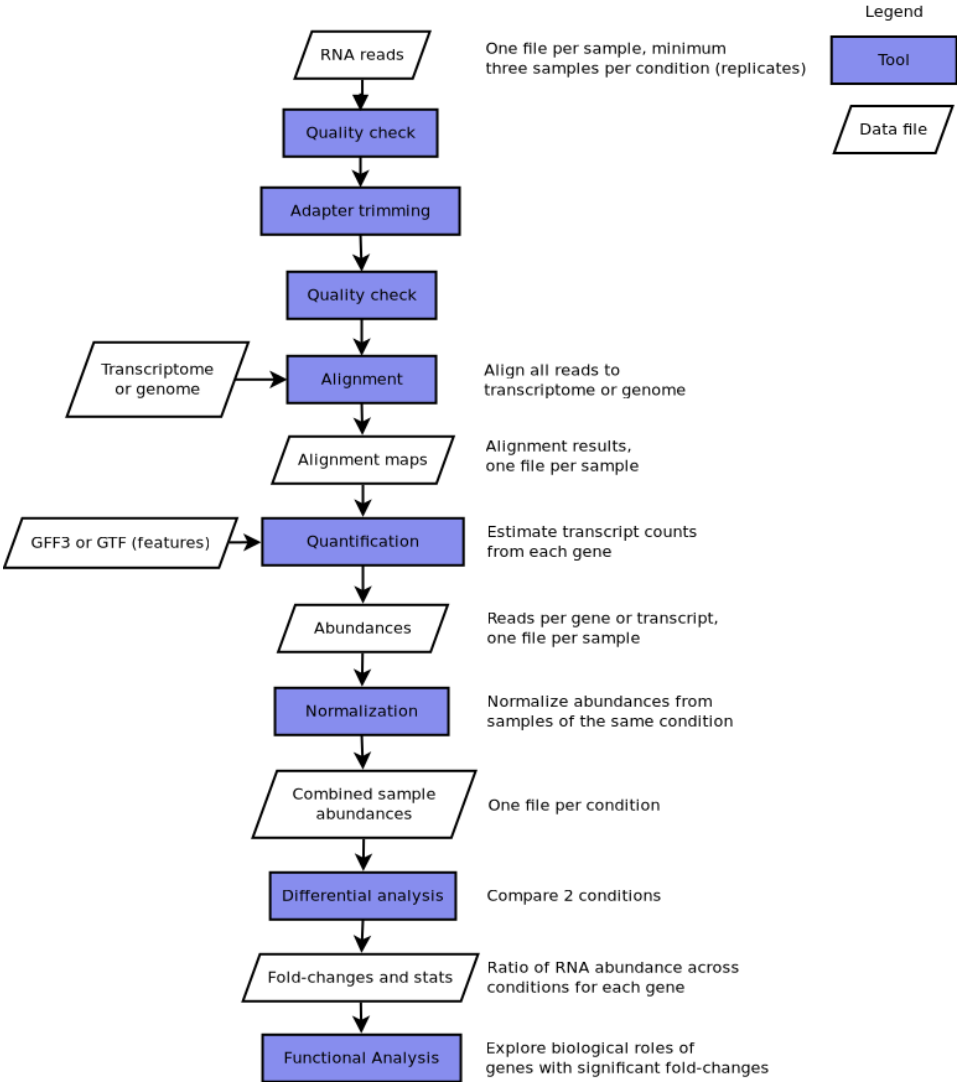

Figure 2.1: Simplified RNA-Seq workflow

2.1.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. How accurate are the RNA abundances we compare in an RNA-Seq differential expression analysis?
2. Is RNA abundance used because it is the best measure of gene activity?
3. What conclusions can we draw from the results of an RNA-Seq differential expression analysis?

2.2 Know your data

One of the core principals of scientific data analysis is *know your data*. In daily life, it is often practical to take a leap of faith. It is not always feasible to gather all the facts necessary to make a fully informed and timely decision. If the consequences of being wrong are not severe (will I like the marinara more than the pesto?), it simply isn't necessary to be rigorous about the decision.

Science, however, is based on thorough verification, never on faith. As scientists, we want to be aware of everything that is happening in our analysis pipeline, and examine the results of each stage before moving on to the next. Choosing an appropriate cancer treatment requires having as much accurate information as possible, and analysing it thoroughly.

As we proceed through our analysis pipeline, we want to know as much about the data as possible. For starters, after receiving the raw FASTQ files from the sequencing center, the first thing we might do is simply have a look at the files:

```
# zmore is a convenience tool, included with most gzip installations
# and equivalent to "gunzip -v file | more"
shell-prompt: zmore cond1-rep1.fastq.gz
@ERR458934.1 DHKW5DQ1:219:D0PT7ACXX:1:1101:1557:2055 length=51
CTCAATTGTCCAAGGCTGGTAAGTTCCCAACCCCAGTTTCTCACAACGATG
+ERR458934.1 DHKW5DQ1:219:D0PT7ACXX:1:1101:1557:2055 length=51
CCCFFFFFHHHHHJJJJJJHIJIIJHIIJJJJJJIICFHIJIJJJJJIIJG
@ERR458934.2 DHKW5DQ1:219:D0PT7ACXX:1:1101:2867:2126 length=51
ACGGTTATTAGAAAGAATGAACAATTCGCAAAAACTCAGAGATTTTTAGTT
+ERR458934.2 DHKW5DQ1:219:D0PT7ACXX:1:1101:2867:2126 length=51
?@@D?DADHHDHHJIIJJJIJJJIGIJIIIIJGJJIIIJJJBGHIJJFGHH
@ERR458934.3 DHKW5DQ1:219:D0PT7ACXX:1:1101:3143:2134 length=51
GAAGTTTTGATCAAGCAAGTTCCAAGACTATTGGGTCCTCAATTGTCCAAG
+ERR458934.3 DHKW5DQ1:219:D0PT7ACXX:1:1101:3143:2134 length=51
@CCFDDFFHHHFHJIGJIIHGHJJHGIIGHGIJIJ?DFHGDHFFHIIEIHI
@ERR458934.4 DHKW5DQ1:219:D0PT7ACXX:1:1101:4636:2212 length=51
CAGTTTCTTCAGCTGGTCTGTAGTCCTTGACAGATGGAGCAAGAATTGGTT
+ERR458934.4 DHKW5DQ1:219:D0PT7ACXX:1:1101:4636:2212 length=51
@@CFFFFFHFHHHJJJEGHJHHJHHIJIIGIIIIJJJJGIJIJFGGHHHDG

[ Output omitted for brevity ]
```

There are also tools available to examine raw FASTQ files and report statistics. One such tool is **fastx-stats**, part of the **biolibc-tools** collection:

```
# blt is the core command of the biolibc-tools package.  It is followed
# by a "subcommand" such as fastx-stats.  Run "blt" with no arguments for
# a list of subcommands of "man blt" for more detailed information.
shell-prompt: blt fastx-stats cond1-rep1.fastq.gz

Filename:           cond1-rep1.fastq.gz
Sequences:          55542528
Bases:              8331379200
Mean-length:        150.00
Standard-deviation: 0.00
Min-length:         150
Max-length:         150
A:                  2209837857 (26.52%)
C:                  1962195390 (23.55%)
G:                  1974345622 (23.70%)
T:                  2184870861 (26.22%)
N:                  129470 (0.00%)
GC:                 3936541012 (47.25%)
```

By looking directly at the data in-hand and comparing what we see to what we expected and what was reported by the sequencing center, we can either catch problems early, or move on to the first stage of the analysis with justified confidence that it will be productive.

2.2.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. Why is it important to know our data?

2.3 Pre-trim quality check with FastQC

This step involves visually inspecting the raw data from the sequencing machine, and using quality control tools that compute summary statistics, and help us visualize them. While RNA-Seq files are vast, tools such as **FastQC** provide a summary view of data quality that help us identify problems in sequencing and determine how to deal with them. FastQC documentation can be found at https://www.bioinformatics.babraham.ac.uk/projects/fastqc/.

For now, we will just have a brief glance at quality control to convey the basic idea of it. In Section 4.7 we take a closer look at quality metrics of some real raw data.

Figure 2.2 shows PHRED score data for each base in a typical FASTQC file. The green zone (PHRED scores above 28, P(error) < 0.00158) is considered high quality by FastQC. The yellow zone (PHRED scores above 20, P(error) < 0.01) indicates acceptable quality for most purposes. The red zone indicates that we probably want to trim these bases off the reads.

The thin red lines indicate the median PHRED score at that base across all reads (which likely number in the millions). The blue line tracks the means, rather than medians. Yellow boxes represent the interquartile range (containing scores from the 25th to the 75th percentile). I.e., the middle 50% of the quality scores are within the yellow box. This means that if the yellow box dips into the red zone, then at least 25% of reads had low quality scores at that position. The whiskers are similar to the yellow boxes, but representing the 10th to 90th percentiles, so 80% of quality scores are within this range.

We can see in this graph that the sequencer was very confident about most of the bases read. However, a small percentage of bases at the 2nd, 3rd, and 4th positions (from the 5' end) have questionable quality scores. Hence, we might choose to trim the first 4 bases from all the reads in order to improve the overall read quality. On the other hand, the mean and median quality scores are good, so if a few incorrect bases are not a problem for our downstream analysis (the analysis steps that follow), then we might choose to keep these bases and have slightly longer reads to align. We don't usually want to throw away any information in our samples that is mostly accurate.

Note We cannot just trim bases 2, 3, and 4, keeping base 1, as this would change the DNA sequence of the read. We must always trim all the way to the beginning or end of a read.

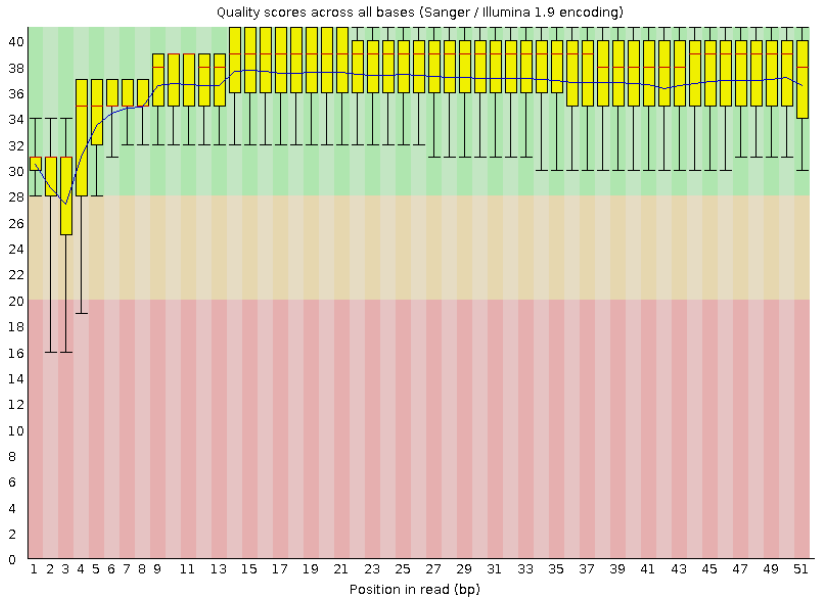

Figure 2.2: Read quality

MultiQC is a tool that aggregates quality metrics from FastQC and some other tools into unified graphs and plots. It is an extremely sophisticated tool that generates browser-based interactive plots that can be exported to images, among other things.

With all this sophistication comes a cost, however. The dreckly MultiQC package depends on 205 other packages at the time of this writing. For this reason, it is not included in the rna-seq meta-package, which has only 82 dependencies without MultiQC, and already takes hours to build and install. MultiQC is considered optional here. If you are interested, you can install the biology/py-multiqc package from dreckly separately. MultiQC *is* included in the FreeBSD rna-seq meta-port, since this is almost always installed from pre-built binary packages, rather than building from source, so adding MultiQC only adds a minute or so to the install time.

2.3.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

---

1. What does the yellow box indicate in a FastQC quality score report?
2. What do we do if bases 46 and 47 have low quality scores, assuming the reads are 50 bases?

2.4 Adapter trimming

Trimming is not strictly necessary for RNA-Seq differential expression analysis [Liao]. However, cleaning up the raw data is straightforward and allows the tools used in later stages, especially read mapping, to perform better. There is no point making all of the subsequent pipeline stages deal with garbage that could have been removed beforehand.

The process of reading DNA or RNA sequences involves appending artificial sequences, called *adapters*, to the ends of each fragment. These adapters are read by the sequencing machine along with the natural DNA/RNA fragment (called the *insert*), and should be removed from the sequence files produced before most types of analyses. Typical short-read sequencers produce fixed-length reads, often between 50 and 150 bases.

The inserts (natural DNA fragments) are created by breaking up DNA with restriction enzymes, a somewhat random process, and therefore vary greatly in length. Fragments may be shorter or longer than the read length. 5' adapters are generally removed automatically at the sequencing center, before they give you your FASTQ files. However, all or part of the 3' adapter may be included in the read sequence if the fragment (including adapters) is shorter than the read length. We use trimming software to locate and remove these 3' adapters. We can also remove other impurities like low-quality reads and poly-A tails at the same time. Figure 2.3 shows a few examples of DNA fragments of various lengths and the resulting reads from each of them.

```
# Fragments with adapter AGATCGGAAGAG attached to both ends

1. AGATCGGAAGAGTGCTGTTACACTTACATGGAGCGGTTCAGCAGGAATGCCGAGATCGGAAGAG
2. AGATCGGAAGAGGTAGCTGATCGATGCTAGCTAGCTAGTCAAGATCGGAAGAG
3. AGATCGGAAGAGTAGTCAGCTAGTGTGATCGATCGAAACGTGTAGCTAGGGGATCTAGTCACCTAGATCGGAAGAG

# Same sequences with spaces between the adapters and insert

1. AGATCGGAAGAG TGCTGTTACACTTACATGGAGCGGTTCAGCAGGAATGCCG AGATCGGAAGAG
2. AGATCGGAAGAG GTAGCTGATCGATGCTAGCTAGCTAGTCA AGATCGGAAGAG
3. AGATCGGAAGAG TAGTCAGCTAGTGTGATCGATCGAAACGTGTAGCTAGGGGATCTAGTCACCT AGATCGGAAGAG

# 50-base reads

1. TGCTGTTACACTTACATGGAGCGGTTCAGCAGGAATGCCGAGATCGGAAG
2. GTAGCTGATCGATGCTAGCTAGCTAGTCAAGATCGGAAGAG??????????
3. TAGTCAGCTAGTGTGATCGATCGAAACGTGTAGCTAGGGGATCTAGTCAC

# Reads with spaces added between insert and adapter contaminant

1. TGCTGTTACACTTACATGGAGCGGTTCAGCAGGAATGCCG AGATCGGAAG
2. GTAGCTGATCGATGCTAGCTAGCTAGTCA AGATCGGAAGAG??????????
3. TAGTCAGCTAGTGTGATCGATCGAAACGTGTAGCTAGGGGATCTAGTCAC
```

Figure 2.3: DNA fragments and reads

Note There is no sequence that is guaranteed not to occur naturally, so "artificial" adapter sequences remaining in the reads cannot be distinguished from natural sequences with 100% certainty. For a typical RNA-Seq analysis, this is not a critical issue. Removal of a few short natural sequences that happen to be the same (or nearly the same) as the adapter will not have much if any effect on the end results. Leaving some adapters in the sequence also won't affect the results that much, but it does create extra work for the tools in subsequent stages.

Adapter contamination is shown by MultiQC, a tool for aggregating FastQC and other quality metrics of many samples, as depicted in Figure 2.4. Here we see that more than 2% of all the reads contain an apparent adapter near the 3' end, and adapters are more likely to occur closer to the 3' end.

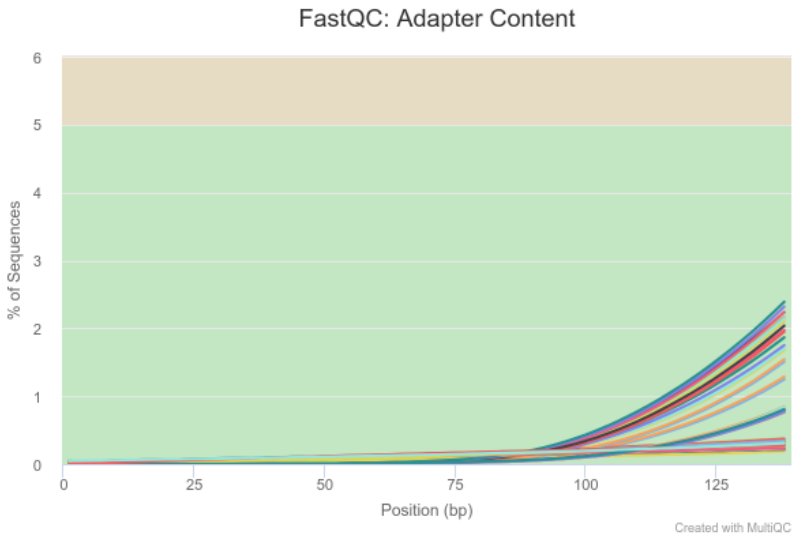

Figure 2.4: Adapter contamination

If, after trimming, the remaining read is too short (typically 30 bases or fewer), we may choose to discard the read entirely. Very short sequences are likely to map to more than one location, making them essentially useless.

When using paired-end data (see Section 1.7), the trimmer must keep the forward and reverse read files synchronized. The Nth read in each file is assumed to be from the same fragment as the Nth read in the other file. Hence, if an entire read is removed from one FASTQ file (e.g. because it is too short after trimming), then the mate must be removed from the other file to keep the mated pairs in the same position of each file.

2.4.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. What is the benefit of trimming our raw data?
2. What are four things trimming aims to remove from our raw sequence data?
3. Show the trimmed version of the following read, assuming the 3' adapter is AGATCGGAAGAG.

```
@
GGGTGATTACACACACCCATTAGCTTTGGGATCGTACGTACTGAAGATCGGAAGAGCTATGCAT
+
:1124??????????????????????????????????????????????????????>>???
```

4. What must trimming software do specially for paired-end data?

2.5 Post-trim quality check

After trimming, we repeat the same quality checks performed before trimming to ensure that trimming had the desired effect. We should see little or no remaining adapter contamination, and any issues evident in other metrics (such as read quality) should generally be eliminated as well. If not, we need to adjust the trimming parameters and trim again.

After trimming, we may see some improvements in read quality as shown in Figure 2.5. Note the difference near the 5' end compared to Figure 2.2.

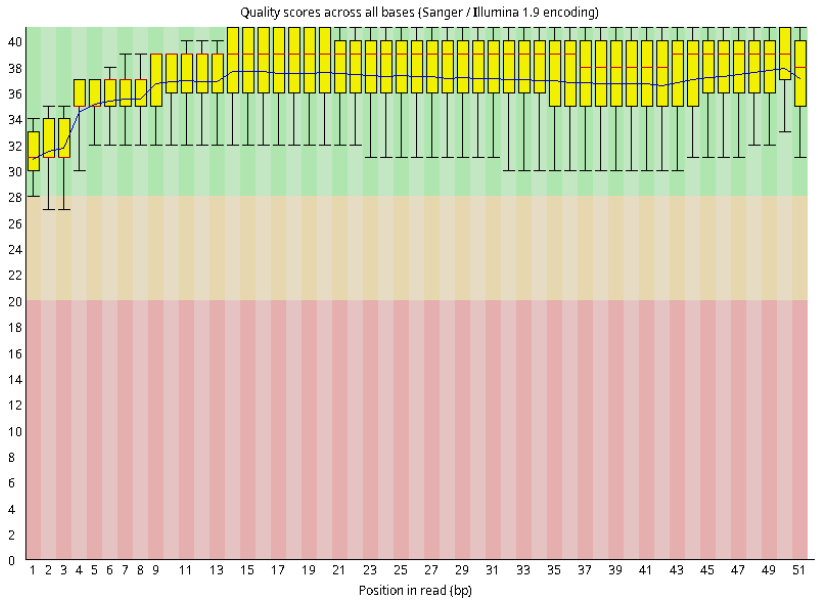

Figure 2.5: Read quality after trimming

MultiQC plots should indicate that problems such as adapter contamination have been mitigated, as shown in Figure 2.6. Note that **MultiQC** will likely still indicate some contamination, since it looks for all possible adapter sequences, rather than just the one we trimmed. Some naturally occurring sequences will occasionally be flagged as adapters, but the percentages shown should be very low.

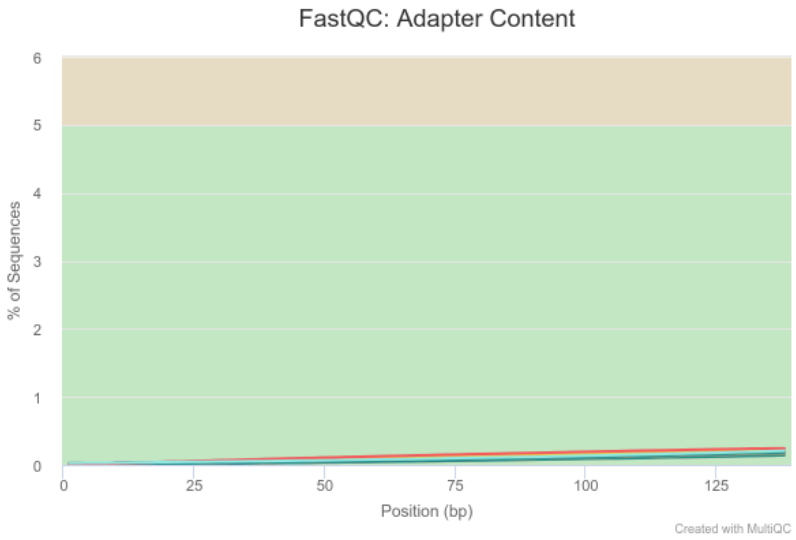

Figure 2.6: Adapter contamination removed

2.5.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. Describe two improvements we should see in the post-trim quality check as compared to pre-trim.
2. Should the post-trim quality check show zero adapter contamination?

2.6 Read mapping (Alignment)

After cleaning up the data, we perform the most computationally expensive step, aligning (mapping) the reads to a *reference*. The reference could be either a *transcriptome*, which contains only the transcribed RNA sequences, or a *genome*, which contains a representative sequence of all DNA. Neither a transcriptome nor a genome reference will match our sequences perfectly, since the reference represents sequences from one hypothetical organism, and no two individuals have identical DNA (not even identical twins, which begin to diverge genetically immediately after the egg cell splits). In reality, the reference may be constructed from multiple individuals, and contain *consensus* bases (the most popular base at that position) where the individuals differ.

The read mapping stage is commonly referred to as "alignment", but this term is vague and should generally not be used. The word alignment could refer to any bioinformatics task that attempts to match two sequences. This includes trimming, which aligns adapter sequences to the reads in order to locate and eliminate them. It includes BLAST searches, which align unknown DNA sequences to a variety of organisms in order to determine their origin.

Aligning RNA to a transcriptome is relatively fast and straightforward, since the transcriptome is very small compared to the genome, and both the RNA reads and the transcriptome contain processed RNA sequences (with introns removed). Hence, they should match up pretty well, except for individual variations between the samples and the reference.

Aligning RNA to a genome requires special tools that can account for RNA processing, since we typically extract RNA from the cytoplasm after it has been processed by removing introns, etc. The genome still contains the introns that have been removed

from the RNA sequences being counted, which makes the alignment more challenging. For this we need a *splice-aware* aligner, i.e. one that can match RNA sequences with large deletions of introns, to a DNA sequence with those introns still present [Krizanovic]. Figure 2.7 shows how a processed RNA sequence with an intron removed aligns to a genome. An aligner that is not splice-aware would consider this a low-quality alignment due to the large deletion relative to the reference, if it can identify such a large deletion at all. A splice-aware aligner would consider this a perfect match, assuming that the insertion matches a known intron in the gene.

```
Spliced RNA sequence:   AAUUGUGGUAGCGAGUCAGCUAUCA

                                  |---------- Intron ----------|
DNA sequence:   AATTGTGGTAGCGAGTCAGTCGATCGATCGTAGCTAGCTAGCTAGCTAGCTATCA
RNA aligned:    AAUUGUGGUAGCGAGUC-----------------------------AGCUAUCA
                                  |-- Deletion vs. reference --|
```

Figure 2.7: Aligning a processed RNA sequence to the genome

When aligning paired-end reads (see Section 1.7 to a genome, the aligner expects that the reverse read (3' end of the fragment) will align not too far downstream of the forward read (5' end of the same fragment). When aligning to a transcriptome, the reverse read should align downstream within the same feature in the transcriptome. If it does, then the alignment is considered *properly paired* or *properly mated*. With high quality reads and a mature reference, we can generally expect more than 80% of the reads to align properly mated, and more than 90% of reads to align individually. If the aligner reports a lower percentage than this, we should stop here and try to identify the issue.

Due mostly to long repetitive sequences and duplications in the genome, many reads will align about equally well to to more than one location. If one alignment is clearly better than the others (has a higher alignment score), it will be tagged in the alignment output as the *primary alignment*.

The output of most aligners is a *sequence alignment map*, stored in the standard *SAM* file format, or one of its compressed equivalents, *BAM (binary alignment map)* or *CRAM (compressed reference-oriented alignment map)*. Raw SAM files are rather "fluffy" and rarely used for this reason. BAM files contain the same information, and are much smaller. This saves a great deal of disk space, and can make reading and writing the files faster, since disk I/O is one of the slowest operations in computers. CRAM files are even smaller, but less convenient than BAM, so they are mostly used for archiving alignments (long-term storage). If you have a desire to work with raw SAM files, then using a filesystem with compression capabilities, such as ZFS, is highly desirable. The *samtools* software suite is a well-developed tool kit for viewing, sorting, and otherwise manipulating SAM, BAM, and CRAM files.

An alternative to traditional alignment is provided by tools such as **Kallisto**, which performs *pseudoalignments* using statistical probability to infer the most likely location without actually matching the entire sequence. **Kallisto** is much faster than traditional aligners, but generally does not achieve quite the same alignment rates. However, if the depth of coverage is good, the loss of some alignments will not significantly affect the final analysis. Whether we successfully align 25 million or 30 million reads to the reference doesn't usually make much difference. **Kallisto** does not produce a sequence alignment map, but instead outputs abundances, as described in Section 2.7. Kallisto works only with a transcriptome reference, not a genome reference.

FastQC also works on SAM/BAM files, and can provide the same metrics about the aligned reads that it does for the raw or trimmed reads from in FASTQ files.

If we have an alignment map (e.g. a BAM file from HISAT2 or STAR), we can view the *pileups* (reads mapping to the same location) using a graphical genome browser such as IGV (Integrated Genome Viewer). IGV loads a reference genome and an alignment map, then displays known features of the genome and pileups from the alignment map. In Figure 2.8, we see some pileups of reads (gray) closely correlated with known exons (blue boxes). For more information, consult the documentation for your version of IGV.

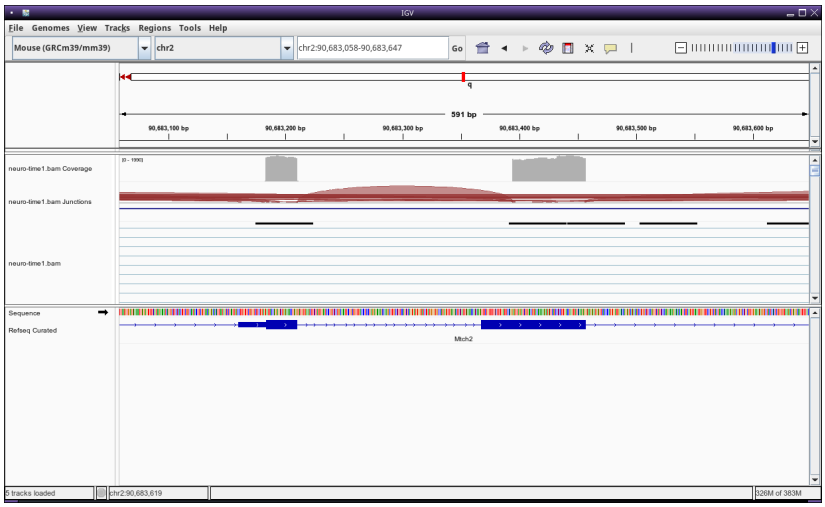

Figure 2.8: IGV

2.6.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. What is read mapping?
2. What is a reference genome or transcriptome?
3. Do reads always match the reference genome or transcriptome perfectly?
4. Are read mapping and alignment the same thing?
5. Is it easier to align RNA reads to a genome or two a transcriptome?
6. What is a properly paired (mated) alignment?
7. What is a primary alignment?
8. Why do most people use BAM instead of SAM file format?
9. What is pseudoalignment and why is it used?
10. How is a genome browser used to investigate mapped reads?

2.7 Quantification (Abundance estimation)

After aligning the RNA sequences with a traditional aligner such as **hisat2** or **STAR**, we estimate transcript abundance by counting how many reads aligned to each gene or transcript. This is not quite as straightforward as counting the aligned reads, because, for one thing, it is common for 10% or more of reads to be *multi-mapped*, i.e. to have more than one good match in the reference.

The **kallisto** pseudoaligner performs its own quantification, so we do not need to perform a separate quantification step after running **kallisto**.

Some popular quantifiers for use on SAM/BAM/CRAM files include **stringtie**, **featureCounts**, and **htseq-count**. All such tools use an alignment map and a list of known features (genes, transcripts, exons, etc.) which is normally provided by a *GTF* (gene transfer format) or *GFF* (general feature format) file. These feature files should generally be downloaded from the same websites

that provided the genome or transcriptome reference in order to ensure compatibility. Files from different sources may report slightly different sets of features.

The grouping of reads aligned to the location of a gene is called a *pileup*. Note that the sequenced fragments are not complete transcripts, but somewhat random fragments thereof, produced by cutting up the transcript with restriction enzymes. Furthermore, the reads are not usually complete fragments, but one or both (if paired-end sequencing was used) ends of each fragment. For example, a transcript might be 4,000 bases long, one of the fragments might be 500 bases, of which we might have 150 bases from each end when doing paired-end sequencing. This leaves 200 bases from the middle of the fragment unaccounted for in the FASTQ files. So, what we're aligning to a gene in the reference genome, or a transcript in the reference transcriptome, is a lot of random bits and pieces of the transcripts. When we view a pileup in IGV, it generally looks very incomplete and messy, like a brick wall with a lot of bricks missing.

Figure 2.9 shows a close-up of a pileup in the center of the image, and IGV's estimate of *coverage* (number of reads mapped to each base) in the top panel. This pileup is actually pretty clean compared to many. We often see more and bigger gaps between the aligned reads. But, it illustrates the problem of accurately determining the exact number of transcripts from a pileup of fragments. Quantifying RNA abundance of a given feature from a messy pileup requires some assumptions and basic math to estimate how many *complete* transcripts were actually in the RNA library that was sequenced.

Figure 2.9: IGV read pileup

Figure 2.10 shows estimated read counts output by Kallisto for a few known transcripts. Note that Kallisto output shows both length (the sum of exon lengths for a transcript) and *effective length*, which is generally defined as the number of start sites that could generate a fragment of this length. This definition is specific to each fragment, so the mean fragment length is often used for simplicity. The effective length is used in computing metrics like TPM (transcripts per million), which give us a better idea about gene activity, normalized for gene length.

target_id	length	eff_length	est_counts	tpm
YPL071C_mRNA	471	282	51	72.5861
YLL050C_mRNA	432	243	840	1387.41
YMR172W_mRNA	2160	1971	121.99	24.8411
YOR185C_mRNA	663	474	74	62.6593

Figure 2.10: Kallisto estimated counts

Note that estimated counts are not necessarily whole numbers, which may seem odd. Just remember that they are an estimate of transcript counts computed from read counts, not counted directly.

2.7.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. What is quantification?
2. What inputs are necessary to perform quantification?
3. Where can we obtain a feature file?
4. What is a pileup?
5. Why do pileups generally look incomplete and messy?

2.8 Normalization

Once quantification is completed for all samples and we have transcript abundances for each gene under each condition and replicate, we normalize the counts across all samples, usually using statistical methods. Each purified RNA sample has a different quantity of total RNA fragments, so we need to adjust them before different samples can be compared to each other. E.g., if biological replicate A has 3 million fragments total, and biological replicate B has 1.5 million, then we would expect to see about twice as many reads aligned to every gene X from replicate A, *assuming that the gene was equally active in both samples*. Normalization attempts to adjust all the reads counts for each sample so that the read counts for gene X above will be equal. In this way, we can, to an extent, distinguish differences due to gene activity from differences due to the total quantity of RNA we were able to extract from the sample.

Normalization for differential expression analysis *does not* utilize common normalization methods that you might be used to seeing in research papers. These include *TPM* (transcripts per kilobase-million), *FPKM* (fragments per kilobase-million), or *RPKM* (reads per kilobase-million). These measures are useful for helping readers visualize the data, but are not suitable for normalizing across replicates for the sake of computing fold-changes [Zhao-norm]. As these standard measures are not relevant to our analysis, we don't spend time on them here. If you're interested in understanding them, however, you can find a concise explanation at https://www.rna-seqblog.com/rpkm-fpkm-and-tpm-clearly-explained/.

The ideal method or normalizing for differential expression utilizes *spike-in controls*, foreign or artificial sequences with a known concentration in every sample. To normalize, we simply multiply all counts by a factor, specific to each sample, to equalize the observed counts of the spike-in control across samples. Spike-ins involve additional cost, however, and more importantly, they are simply not present in most data that we are likely to download from previous studies.

Statistical methods such as *median ratio normalization* (MRN) generally work well enough, however, so the absence of spike-ins is not a major disadvantage.

Normalized counts for replicates of the same condition (e.g. wild-type vs mutant) are then pooled to produce a single abundance estimate for that condition. This normalized count can then be compared to normalized counts for other conditions.

2.8.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. Why do we need to normalize abundance estimates obtained in the quantification stage?
2. What is the best way to ensure accurate normalization across samples? Why don't we always use this approach?

2.9 Differential expression analysis

After obtaining normalized abundance estimates for each gene/transcript under each condition (wild-type vs mutant, normal vs stressed, etc), we can compare any two conditions to each other. The basic measure reported is *fold-change*, or *FC*. This is simply the ratio of two abundance estimates:

$FC_{1,2}$ = $abundance_{condition2}$ / $abundance_{condition1}$

A fold-change of 1.0 indicates no difference in RNA abundance between the conditions. A fold change > 1.0 indicates up-regulation in condition 2 vs condition 1. A fold-change between 0 and 1 indicates down-regulation. Many researchers will report log(FC) instead, since positive and negative logs are easier to compare to each other than a fraction and its inverse, as shown in Table 2.1. How many of us would recognize immediately that 0.4 is the exact inverse of 2.5?

Fold-change	**log(fold-change)**
2.5	.916
0.4 (1/2.5)	-.916

Table 2.1: Fold-change vs log(fold-change)

A fold-change, in theory, indicates whether a gene is up- or down-regulated in one condition or another, and by how much. Bear in mind, however, that it is based on read counts, which are a crude estimate of RNA abundance, which in turn is a crude estimate of gene expression rate, which in turn is a crude indicator of a biologically meaningful difference. Unless a fold-change computed from read counts is dramatic, it should not generally be taken as a clear indication of an important change in the organism.

Determining which fold-changes are "significant" is a dubious task. First, read counts are not always an accurate reflection of gene activity, due to many variables in the extraction and sequencing process. For example, Figure 2.11 shows the *technical variation* from publicly available yeast samples available from the sequence read archive (SRA) at https://www.ncbi.nlm.nih.gov/-bioproject/PRJEB5348. Specifically, each count is from a *technical replicate*, a partition of *the same biological sample* that was sequenced separately, but on the same sequencing machine at the same time. We can see that even technical replicates often show high variability in read counts. For example, the estimated counts for feature YLL050C_mRNA range from 194 to 282, even though these are subsamples from the same biological sample. Does this indicate poor pipetting technique? Probably not, as there a many variables beyond the control of the people handling the solutions and running the sequencer.

```
===> Results/Tech-reps/kallisto-quant/ERR458493/abundance.tsv
target_id       length  eff_length  est_counts   tpm
YPL071C_mRNA    471     282         14           35.3946
YLL050C_mRNA    432     243         240          704.147
YMR172W_mRNA    2160    1971        43.6935      15.8048
YOR185C_mRNA    663     474         44           66.1809

===> Results/Tech-reps/kallisto-quant/ERR458494/abundance.tsv
target_id       length  eff_length  est_counts   tpm
YPL071C_mRNA    471     282         17           43.3106
YLL050C_mRNA    432     243         261          771.664
YMR172W_mRNA    2160    1971        39.4246      14.3706
YOR185C_mRNA    663     474         41           62.1441

===> Results/Tech-reps/kallisto-quant/ERR458495/abundance.tsv
target_id       length  eff_length  est_counts   tpm
YPL071C_mRNA    471     282         13           33.329
YLL050C_mRNA    432     243         249          740.835
YMR172W_mRNA    2160    1971        54           19.8078
YOR185C_mRNA    663     474         43           65.5871

===> Results/Tech-reps/kallisto-quant/ERR458496/abundance.tsv
target_id       length  eff_length  est_counts   tpm
YPL071C_mRNA    471     282         13           35.9038
YLL050C_mRNA    432     243         245          785.246
YMR172W_mRNA    2160    1971        39.1726      15.479
YOR185C_mRNA    663     474         37           60.7952

===> Results/Tech-reps/kallisto-quant/ERR458497/abundance.tsv
target_id       length  eff_length  est_counts   tpm
YPL071C_mRNA    471     282         13           42.0872
YLL050C_mRNA    432     243         194          728.872
YMR172W_mRNA    2160    1971        31.4292      14.558
YOR185C_mRNA    663     474         29           55.8567

===> Results/Tech-reps/kallisto-quant/ERR458498/abundance.tsv
target_id       length  eff_length  est_counts   tpm
YPL071C_mRNA    471     282         17           55.287
YLL050C_mRNA    432     243         206          777.471
YMR172W_mRNA    2160    1971        22.5807      10.5069
YOR185C_mRNA    663     474         37           71.589

===> Results/Tech-reps/kallisto-quant/ERR458499/abundance.tsv
target_id       length  eff_length  est_counts   tpm
YPL071C_mRNA    471     282         14           34.8473
YLL050C_mRNA    432     243         282          814.579
YMR172W_mRNA    2160    1971        44.8517      15.9728
YOR185C_mRNA    663     474         53           78.4852
```

Figure 2.11: Variance among technical replicates

There are also many biological variables to consider. Doubling the transcription rate, which in theory should produce a fold-change of about 2, may be highly significant for genes where a small increase in RNA transcripts (and presumably the protein product) is amplified by a metabolic pathway, but almost meaningless for others where a doubling the abundance of this particular RNA has little effect on the cell.

Counts from different biological replicates often vary immensely due to individual differences among the organisms, uncontrollable differences in the state of each individual (maybe one had an unknown viral infection at the time of sampling), and timing (individuals may have been in different stages of the cell cycle on average across the cells used).

A P-value (the probability that a change at least as extreme as the one reported would occur by chance) is typically computed for each fold-change, considering abundances from all of the replicates separately. P-value computations use standard statistical methods that consider the count ratio and the consistency of the counts across replicates. Hence, we look for fold-changes with very low P-values, typically 0.05 or less. Note that P-values cannot be computed without multiple replicates, and more replicates result in more reliable P-values, in the same way sample size affects confidence in any statistic.

Caution Calculation of P-values does not incorporate any knowledge of biology. The P-value only tells us the likelihood that this particular set of estimated counts would occur at random. A low P-value in no way indicates that a significant biological change has occurred.

Hence, P-value provide only one circumstantial piece of evidence regarding the significance of a fold-change. We cannot simply use the standard cutoff of P <= 0.05 to decide whether a change in expression is meaningful. We must consider biological factors related to each gene along with the P-values, and allow for a wide margin of error in order to identify which changes are most significant to our study. As there may be thousands of genes with potentially significant changes, it is tempting to look for an easy selection method, but doing so will lead to many errors (false positives and false negatives) in the analysis. P-values are used as a first step in narrowing the list of significantly up- or down-regulated genes, but should not be taken too seriously.

Figure 2.12 shows an example of output from **fasda fold-change**, which includes mean normalized counts (MNC1, MNC2), standard deviation of counts within replicates, fold-changes, and P-values for each of 4 genes. All of this information, and more if available, should be considered when deciding which changes in read counts are significant.

Feature	MNC1	MNC2	SD/C1	SD/C2	FC 1-2	log2(FC)	P-val
YPL071C_mRNA	29.6	31.1	0.3	0.2	1.05	0.07	0.72611
YLL050C_mRNA	399.5	543.8	0.2	0.2	1.36	0.44	0.02857
YMR172W_mRNA	60.6	83.3	0.2	0.2	1.38	0.46	0.02167
YOR185C_mRNA	57.1	56.1	0.3	0.2	0.98	-0.03	0.92562

Figure 2.12: Fold-changes and statistics

Given the many biological and technical variables that cast doubt on our fold-changes and P-values, we might end up thinking "what's the point?". While the facts clearly indicate that the results of our analysis are very imprecise, there is a bright side. Like many biological experiments, differential analysis produces a vast amount of information, most of which is unclear, but some of which will lead to new discoveries. Even when working with unreliable data, when there is enough of it, we *will* get lucky in some cases if we are persistent. Using P-values to guide is is sketchy, but a whole lot better than picking genes to investigate at random. Explore the results of the analysis with persistence, and eventually you will find something of interest that can be experimentally verified. You will also discover many dead-ends, but that's part of science, and you may learn something serendipitously from those misadventures as well.

2.9.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. What is a fold-change?
2. Why do we often use log(FC) instead of FC?
3. How reliable are fold-changes for determining a biologically significant event?
4. What is technical variation?
5. Does a fold-change of 3.6 indicate a significant biological change?

6. What is a P-value?
7. Does a low P-value indicate a biologically significant event?
8. If P-values are not reliable, why do we use them?

2.10 Functional analysis

Computing fold-changes and P-values is the last predominantly computational step of a differential expression analysis. From here, we take our results and move on the *functional analysis*, where we try to discover the functions and interrelations of the genes or transcripts that are apparently differentially expressed. Functional analysis is introduced in Section 4.19, though it is not covered in detail in this text.

2.10.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

---

1. No practice questions for this section.

Chapter 3

Installing the RNA-Seq software

3.1 The status quo: A perilous obstacle course

One of the most cost-effective ways to increase research throughput is to simply stop placing unnecessary hurdles in front of ourselves. Software installation is one of those hurdles, as I have witnessed in nearly 20 years providing I.T. support for scientific research. While installation of any open source software package *can* and *should* be trivial, software installation is still, unfortunately, a major hurdle for many scientists attempting to perform data analysis. Many times I've seen researchers struggle for days or weeks just to install one program, often giving up in the end.

Easy installation of most open source scientific software is not a Utopian dream. Not only is it perfectly feasible, but doing so would dramatically *decrease* the total time and resources spent on software deployment. Currently, thousands of scientists often suffer through the duplicate wasted effort attempting to deploy the same poorly-supported software on their computers. If one person invested the time to package it properly (which often takes less time than trying to install it once by more primitive methods) then the struggles of thousands of others would be completely eliminated, and thousands of wasted [wo]man-hours could be used for more productive endeavors instead. The only barriers to making this dream a reality are awareness and human inertia.

I started my career in scientific computing support performing many manual *cave-man* software installations (downloading tarballs of the software source code, and following poorly-written and incomplete instructions for building and installing the software). Nobody should be installing software this way in the 21st century, but there was no other way at the time, and still isn't for some software. Given the variety of Linux distributions and SGI IRIX systems we had, all requiring different patches, these cave-man installs occupied a significant portion of my time. This was in stark contrast to the effortless installation of most of the non-scientific software we used, such as the **nedit** text editor, **gcc** compilers, etc. via *package managers* such as FreeBSD ports. It quickly occurred to me how nice it would be if I could deploy *all* of our software using a package manager. This would free up a lot more of my time to work on improvements to our systems, work directly with researchers on novel problems, etc.

So, I looked into creating FreeBSD ports of my own for our major scientific tools. What I found was that creating a package often required less effort than doing *one* cave-man install, and vastly less time than doing cave-man installs on all of our 30 or so Unix workstations. Once a package was created, I could then deploy the software on each of our workstations with one simple command in a matter of minutes. Better yet, by creating a *meta-package*, that installs a list of other packages, I could deploy all the tools used by our research group on every workstation with a single command. Having every program working reliably on every workstation was a dream come true for our researchers.

Installing all of the tools necessary for an RNA-Seq analysis has traditionally involved a wide variety of installation methods, including multiple different package managers, downloading binaries directly from developer websites, various types of containers, and cave-man installs.

Downloading binaries (precompiled programs) directly from developer sites is a dangerous habit from a computer security standpoint. Binaries can horbor malware such as computer viruses, Trojan horses, etc. Trusting every scientific software developer's website to protect us from malware is not a realistic expectation. Many of these sites are maintained by scientists who lack the time and/or skills to maintain strong computer security. This is why Apple's App Store and Google play exist. These services perform standardized security checks in order to minimize the risk of users installing malware on their devices. They eliminate the need to trust a wide variety of websites. Instead, we need only trust these two to gather software from the rest and ensure that

it is free of harmful malware before we download it to our own devices. Scientific software developer websites vary greatly in their level of security, so trusting all of them implicitly is asking for trouble.

Using a centralized package manager, such as Debian packages, dreckly, FreeBSD ports, etc. provides some of the same security benefits. All of the packages are built on and distributed from servers managed by the same project, which in turn is run by people with much more knowledge of computer security than the average scientific software developer.

Downloading developer-built binaries also leads to compatibility issues. Developers usually only provide prebuilt binaries for one or a few platforms. They will not provide binaries for every version of macOS or for every one of the dozens of available GNU/Linux distributions, for example. They might only provide binaries for x86-based processors, and not for ARM, Power, etc. A binary built on one version of macOS or Ubuntu Linux will not generally work on older macOS versions or RHEL Linux. Even worse, developers don't all support the same platforms, so you may need to maintain multiple computers in order to run all the developer-supplied binaries you need.

Package managers, on the other hand, provide packages for all actively maintained versions of the operating systems they support, and all supported processors. Software developers will generally serve their users and themselves better by making their software easy to build on any POSIX platform, and leaving the builds and installation to package managers.

R-based software installations generally use installation tools within the R environment, namely **install.packages()** for installing packages from the CRAN package collection, or **BiocManager::install()** for installing from the Bioconductor package collection. Neither of these tools manages *all* of the prerequisite software for the packages it installs. Instead, they rely on other package managers outside of R to provide some dependencies. This often leads to failed package builds, and if the build does succeed, the situation is a time bomb. When dependencies installed outside of R are removed or upgraded, the R packages installed by **install.packages()** or **BiocManager::Install()** become broken. Below is a typical error message you might encounter when such breakage occurs. In this case, the problem was that the "icu" library was upgraded outside of R.

```
Error: package or namespace load failed for 'stringr' in dyn.load(file, DLLpath = DLLpath,  ↩
    ...):
unable to load shared object '/usr/home/bacon/R/library/stringi/libs/stringi.so':
Shared object "libicui18n.so.71" not found, required by "stringi.so"
```

Dealing with errors like this one requires extensive I.T. skills that most biologists do not have, and often results in significant delays to research. The research process is slow enough already, without adding avoidable roadblocks like this one.

Caution

If you run R from an unprivileged dreckly package manager installation (one that was not installed at the root user), you must be careful about conflicts between dreckly R packages and those installed from within R, e.g. using **install.packages()**. The default installation prefix for R packages under the dreckly tree will be writable, so **install.packages()**, etc. could overwrite files installed by dreckly packages or vice versa.
To avoid this problem, add **.libPaths("~/R")** to `~/.Rprofile`, and run **mkdir -p ~/R**, so **install.packages()**, etc. install things there instead of clobbering files under the dreckly prefix.

3.1.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. How hard should it be to install an open source software package?
2. Why is it dangerous to download binaries from individual developer websites?
3. What limitations will we encounter when using binaries from developer websites?
4. What is a major issue with software installed within the R environment using `install.packages()`?

3.2 A secure and reliable approach: Package managers

There are many ways to install the necessary software for an RNA-Seq differential expression analysis. As mentioned earlier many require advanced I.T. skills, and even if you have them, you will waste a significant amount of your time. To remedy this situation, I, leveraging the work of hundreds of other package maintainers, automated the installation of all the necessary software for our analysis, on any Unix-like system you are likely to be using, via one of two package managers:

- FreeBSD ports is the package manager for FreeBSD, GhostBSD, and effectively for Dragonfly BSD, which uses a derivative called "dports". Using FreeBSD ports, you can install all of the necessary software using prebuilt packages in a few minutes. At the time of this writing, there are 33,268 ports in the FreeBSD ports collection, including 2,389 in the scientific categories. These numbers increase almost every day.
- Dreckly is a package manager committed to supporting *all* Unix-like systems. It works well on macOS and most GNU/Linux distributions. This installation is just as easy for you as using pre-built packages from FreeBSD ports, though it will take hours rather than minutes, because it builds all of the packages from scratch. You need only start the installation, and dreckly will do the rest for you automatically. At the time of this writing, there are 19,815 packages in the dreckly collection, including 850 in the scientific categories. These numbers increase almost every day.

Package managers vary greatly in their capabilities, portability, and security. Conda is a popular tool for installing scientific software, but many users are unaware of the disparity between conda *channels* (essentially separate package repositories). The Conda "main" channel includes standard quality and security checks, while "community" channels such as Bioconda have a lower bar for quality and security. AUR is a community-driven package manager for Arch Linux, which serves as a testing ground for the official Arch package manager, Pacman. AUR contains a huge number of packages, including many of the those used for the analysis presented here. Debian packages (only for Debian Linux and derivatives like Ubuntu) contains the largest "official", fully-tested collection of packages, followed by FreeBSD ports, GNU GUIX, Fedora packages, etc. However, at the time of this writing, the Debian package collection does not provide all the tools necessary for our analysis.

You can find much more information about package managers in The Research Computing User's Guide and at https://repology.org/-.

If you already have access to a Unix system, such as a Mac or a Linux PC, you can use the dreckly package manager, as described in Section 3.3. If you do not have access to a Unix system, then Section 3.4 will show you how to install GhostBSD, a free derivative of FreeBSD designed for Unix NooBs (pronounced "newbies"). FreeBSD itself works very well for scientific computing (I use it for most of my software development and bioinformatics work), but it requires some Unix skills to install and manage. FreeBSD is primarily used by Unix developers or as a server operating system. For example, the Netflix content delivery network uses FreeBSD servers. If you are interested in trying FreeBSD, the desktop-installer package will help you quickly set up a FreeBSD desktop system.

You can run GhostBSD directly on a PC if you have one to dedicate, or under a virtual machine on most computers running popular operating systems such as BSD, Linux, macOS, or Windows. Once GhostBSD is installed, you can install all of the software needed for our analysis pipeline in a few minutes using *Software Station*, a graphical user interface to the FreeBSD ports collection. To simplify installation even further, *meta-packages* (packages that simply install a group of related packages) were created to automatically install all of the tools needed for an RNA-Seq, ChIP-Seq, or ATAC-Seq analysis in one step. Using the package manager to install the single meta-package "rna-seq" will automatically install everything you need for the analysis presented in this text.

3.2.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. What is the major advantage of the dreckly package manager?
2. What is the quickest way for people who only have an MS Windows machine to gain access to Unix and all the software needed for an RNA-Seq analysis?

3.3 Dreckly package manager

As stated earlier, the dreckly package manager can be used to install software on virtually any Unix-compatible operating system. For bioinformaticians, this will most likely mean Linux or macOS. Dreckly is a recent derivative of pkgsrc, which was created for NetBSD, but is designed to function on any Unix-compatible system. However, the pkgsrc community is not as committed to supporting non-NetBSD platforms as is dreckly. Those running FreeBSD, or a derivative such as GhostBSD, could in theory use dreckly as well, though using the native FreeBSD ports system is much faster and easier.

Dreckly has some major advantages over most other package managers:

- It works on virtually any Unix-compatible platform, including most GNU/Linux distributions, and macOS, which are extensively used in bioinformatics and other scientific computing. Hence, we can install the *exact same software* on our Macs and various Linux systems with minimal effort. This eliminates barriers to collaboration and reproducibility, and gives us the freedom to choose our operating system based on its technical merits (performance, reliability, etc.) rather than convenience of installing the particular software we need.

  Note Many package managers claim to be portable, but at the time of this writing, dreckly is the *only* one that is fully committed to supporting a wide variety of Unix platforms including BSD, GNU/Linux, macOS, and SunOS.

  Part of what makes dreckly so portable is its ability to build packages from scratch on your own computer just as easily as downloading and installing a binary (prebuilt) package. This enables dreckly to function smoothly on the wide variety of Linux distributions that are not binary-compatible with each other (https://en.wikipedia.org/wiki/List_of_Linux_distributions). For example, prebuilt packages from Ubuntu will not work on RHEL, but dreckly will generally work fine on both, since it builds its own packages right on your computer. Building the packages locally takes longer for the computer, but demands no more of your time. There are binary packages available from dreckly for some platforms, but building packages for every version of every Unix-compatible operating system is not feasible. We focus here on building from source so that the instructions will apply to every reader.

- Dreckly does not require administrator rights. If you have a Unix-compatible computer, such as a Mac or Linux machine, that is managed by your organization, you can still set up dreckly and install packages with little or no help from your I.T. department.

  You may need help from your I.T. department just to install the compiler and other command-line development tools needed by dreckly. On macOS, you would simply ask your I.T. guy to make sure the Java development kit is installed, and run the following command to install the development tools:

```
shell-prompt: macOS: xcode-select --install
```

- Dreckly can be configured to install its own modern compilers and use them to build all dreckly packages (except the compiler itself). This facilitates building modern open source software on enterprise Linux systems, which ship with older compilers, libraries, and other tools.

This section shows you how to easily "bootstrap" dreckly (install the dreckly system) and then use dreckly to install the tools necessary for our RNA-Seq pipeline. To install dreckly, first open a terminal emulator. If you don't know what that is, start reading part 1 of The Research Computing User's Guide, and come back here when it's clear. It won't take long to get that far. Then run the following commands:

```
# Download the auto-dreckly-setup script from GitHub.  This makes it
# easy to bootstrap the dreckly system.
shell-prompt: curl -O https://raw.githubusercontent.com/outpaddling/auto-admin/refs/heads/ ↩
    master/User-scripts/auto-dreckly-setup

# Make the script executable (Unix has optional read, write, and execute
# permissions on every file)
shell-prompt: chmod a+x auto-dreckly-setup

# Install (bootstrap) the dreckly system
shell-prompt: ./auto-dreckly-setup
```

Once **auto-dreckly-setup** is running, you can accept default answers to most questions.

Note If you are running on an enterprise Linux system, such as Alma, CentOS, RHEL, or Rocky Linux, or an outdated operating system using an old GCC compiler, choose a modern GCC compiler when asked. This will ensure that most dreckly packages will build successfully. The 2nd newest GCC shown in the list is usually the best bet. The newest GCC will likely cause more build failures, since new language standards are not always 100% backward-compatible, and it takes time for software developers to fix compatibility issues.

When **auto-dreckly-setup** completes, it should automatically open the **auto-software-manager** menu, as shown in Figure 3.1. From here, you can install any dreckly package by providing the category and name. Select the "Install port/package from source" option and enter "biology/rna-seq".

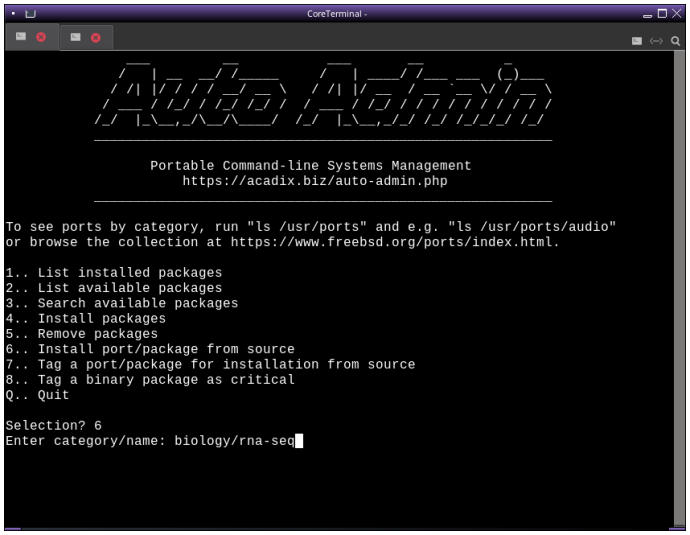

Figure 3.1: auto-software-manager

To use your dreckly installation in the future, you will need to configure your shell environment. This is explained in The Research Computing User's Guide. You do not, however, need to understand the details right now. Simple run the command shown below.

```
# We need to add the dreckly system to your shell's PATH, so it
# can find the appropriate commands.  Do this once in each new
# terminal window.  You can also add this to your shell's startup
# script to make it happen automatically when you open a terminal.
# The The Research Computing User's Guide for instructions.
# dreckly.sh should work for bash, dash, and most other common shells
# Change this to dreckly.csh if you are using csh or tcsh
shell-prompt: source ~/Dreckly/pkg/etc/dreckly.sh
```

You need only do this once for each terminal window, and you can configure it to be done automatically when you open a terminal, following the instructions in The Research Computing User's Guide.

The rna-seq package is a "meta-package" which does not install anything itself, but lists numerous other packages required for RNA-Seq differential expression analysis as dependencies. At the time of this writing, it installs a total of 82 packages. Imagine yourself installing all this software manually, using a wide variety of installation methods, many of which don't work well and require advanced I.T. skills to complete. That's the norm in scientific computing, unfortunately. Instead, you simply run **auto-dreckly-setup** once, install one meta-package, and you're ready to start your analysis.

Installation of the rna-seq meta-package may take several hours, depending how fast your computer is and how many CPUs it has, but it should complete on its own without any intervention from you. If it fails for any reason, please report the problem

at https://github.com/drecklypkg/dreckly/issues. When the installation of the rna-seq meta-package completes, your terminal window should look like Figure 3.2.

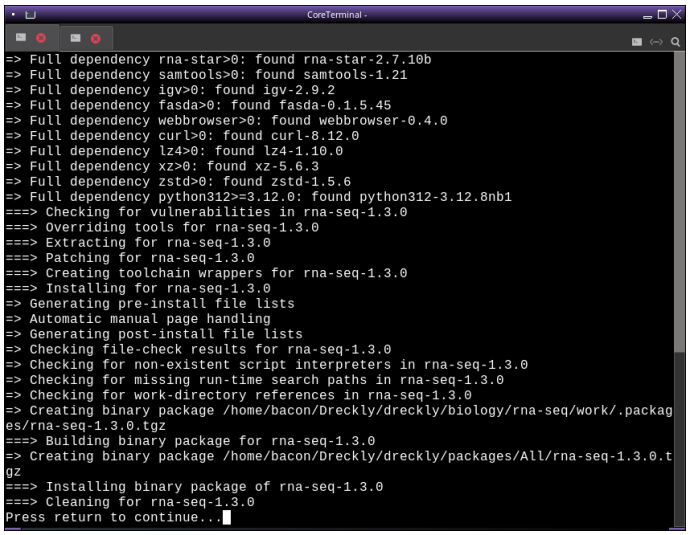

Figure 3.2: Completion of rna-seq install

When the installation completes, you can exit **auto-software-manager** and return to the Unix shell by typing 'q'. You can also install dreckly packages from the shell, rather than use the **auto-software-manager** menu. Assuming you accepted the default location when running **auto-dreckly-setup** and sourced `dreckly.sh` or `dreckly.csh` script as shown above, you can install **git** via dreckly as shown below:

```
# Source dreckly.sh (or dreckly.csh) if you haven't already done so
# in this terminal window.
shell-prompt: source ~/Dreckly/pkg/etc/dreckly.sh

# Install git for downloading scripts, etc. related to this text
shell-prompt: cd ~/Dreckly/dreckly/devel/git
shell-prompt: sbmake install
```

The **git** command will be needed to download the scripts used for the example analysis in Chapter 4.

At any time in the future, you can upgrade all of your previously installed dreckly packages using one simple command, assuming the sysutils/auto-admin package is installed. The **auto-dreckly-setup** script installs auto-admin automatically.

```
# Source dreckly.sh (or dreckly.csh) if you haven't already done so
# in this terminal window.
shell-prompt: source ~/Dreckly/pkg/etc/dreckly.sh

# Upgrade all installed packages
shell-prompt: auto-dreckly-update --defaults
```

Run **auto-software-manager** at any time to install more packages, list or search available packages, etc.

```
# Source dreckly.sh (or dreckly.csh) if you haven't already done so
# in this terminal window.
shell-prompt: source ~/Dreckly/pkg/etc/dreckly.sh

# Start the software manager menu
shell-prompt: auto-software-manager
```

3.3.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. What are three advantages of the dreckly package manager.
2. What is necessary to use the dreckly system after it has been installed?
3. How can a dreckly user update all of the software installed by dreckly?

3.4 GhostBSD

3.4.1 Background

GhostBSD is a FreeBSD derivative designed to be as easy as possible to install and manage, even if you know nothing about Unix. It can be installed very quickly, either directly on a PC, or under a virtual machine monitor such as VirtualBox, which is also free, and well-integrated with GhostBSD.

This section is mainly aimed at Windows users, as it provides them the quickest and easiest way to get access to a Unix system and install all of the software needed for our RNA-Seq pipeline. If you already have access to a Unix system, such as Linux or macOS, using the dreckly package manager on your existing system may be preferable. However, you can run GhostBSD in a virtual machine on any computer with a graphical display and a virtual machine monitor such as VirtualBox. So, if you have a Mac or a Linux machine, the choice between using dreckly package manager and a virtualized GhostBSD installation is entirely yours.

If you have a PC to dedicate to RNA-Seq analysis, you can download and install GhostBSD on it by putting the GhostBSD installer image on a USB stick and booting from it. See the GhostBSD website for instructions. This text uses the primary GhostBSD image with the "Mate" (pronounced like "mah-tay") desktop environment. If you do not have a dedicated PC, but have a reasonably powerful computer running BSD, Linux, macOS, or Windows, then you can download and install VirtualBox, following the instructions at https://www.virtualbox.org/. Then follow the brief instructions in Section 3.4.2 to create a new virtual machine. In either case, Section 3.4.3 explains how to install the GhostBSD operating system.

There are many free operating systems you could choose for this purpose. GhostBSD is recommended for the following reasons:

- No other operating system is easier to install and maintain than GhostBSD. It is an ideal first operating system for those with minimal I.T. skills.
- All of the software needed to complete the analysis described in this text can be installed in a few minutes via the graphical Software Station application. This is the easiest way possible to install all the necessary software.
- GhostBSD uses the advanced ZFS filesystem, with built-in compression, which saves a lot of disk space when working with raw sequence files such as FASTA, FASTQ, and SAM.
- GhostBSD is an augmented form of FreeBSD, and is therefore extremely fast and reliable. The vast majority of FreeBSD servers and desktop PCs I have managed over the past 30 years have *never* crashed. Of the few crashes that did occur, most were caused by a hardware failure. I have experienced some issues with laptops related to suspend/resume (putting the laptop to sleep when closed), but those have been infrequent (less than one per year), and the bugs causing them usually get fixed within a matter of weeks.

You could also install a Linux system and then use dreckly to install the software. However, this would be less convenient than GhostBSD + Software Station, and it would take hours to complete the setup rather than minutes. Linux systems also tend to have higher resource requirements than FreeBSD, which can be problematic for a virtual machine guest, as the virtual machine shares limited resources with the host on which it runs. Lastly, Linux systems are not quite as reliable as FreeBSD (based on my 20 years of experience professionally managing Linux servers and desktop workstations), and typically crash about once a month. This is unlikely to be an issue for running the example analysis in this text, though. It's more of a concern for someone who manages many Linux installations that people depend on. If you would like to try Linux, it should work fine for this purpose. My top recommendation would be Debian, which is easy to install and manage, more stable than Ubuntu in my experience, and offers a huge collection of Debian packages installable via the native **apt**.

3.4.2 VirtualBox

VirtualBox is a free and open source virtual machine monitor (VMM) with excellent support for running graphical operating systems. A VMM is essentially a program that pretends to be a computer. You can install an entire operating system under a virtual machine the same way you install them on real hardware. There are several free VMMs available, but none are as easy to use as VirtualBox. VirtualBox is more comparable to commercial products such as Parallels and VMWare. VirtualBox runs on most popular operating systems, including FreeBSD, Linux, macOS, and Windows. The system running VirtualBox is called the *host*. The list of operating systems that can run inside virtual machines *under* VirtualBox (called *guest* systems) is vast. Figure 3.3 shows GhostBSD running as a guest on a FreeBSD host.

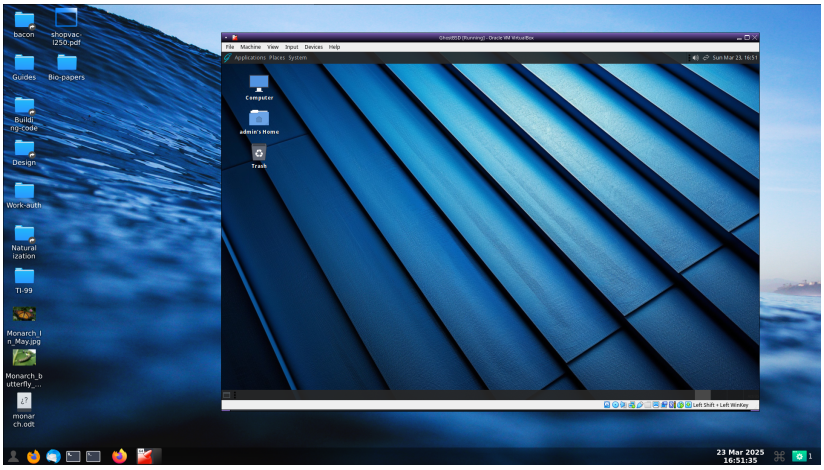

Figure 3.3: GhostBSD as a guest under FreeBSD

To install GhostBSD under VirtualBox, first install VirtualBox, following the instructions at https://www.virtualbox.org/. Then download a GhostBSD ISO image from the GhostBSD website, launch the VirtualBox GUI (graphical user interface), click "New" to create a new virtual machine, and enter the info as shown in Figure 3.4.

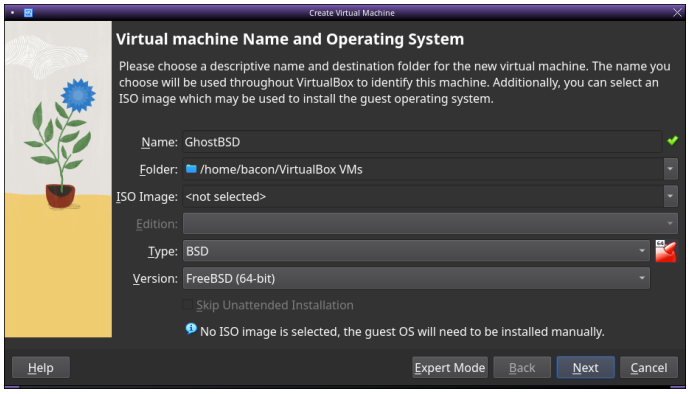

Figure 3.4: VirtualBox VM name and OS

Then click "Next", and change the memory size to at least 4 GiB (4096 MB) as shown in Figure 3.5. This is a requirement of the GhostBSD installer. You can reduce it after installing GhostBSD, but many applications will require this much memory anyway, so it's best to leave it. Set the number of virtual CPUs to whatever you like. It's OK to use all of the available CPUs on your machine (end of the green line in the slide bar), unless you want to reserve some just for the host operating system.

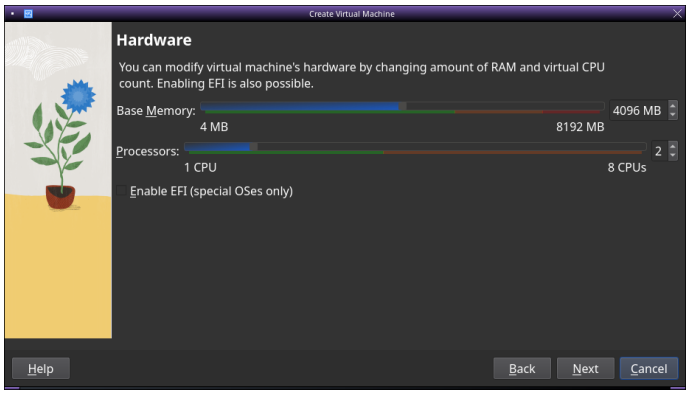

Figure 3.5: VirtualBox memory size and CPUS

Accept default answers until VirtualBox asks you for the disk size, and adjust as shown in Figure 3.6. If you're just doing the Yeast example in this text, 30 GB should be plenty. If you plan to use this VM to analyze more complex organisms, you'll need a lot more, possibly over a terabyte.

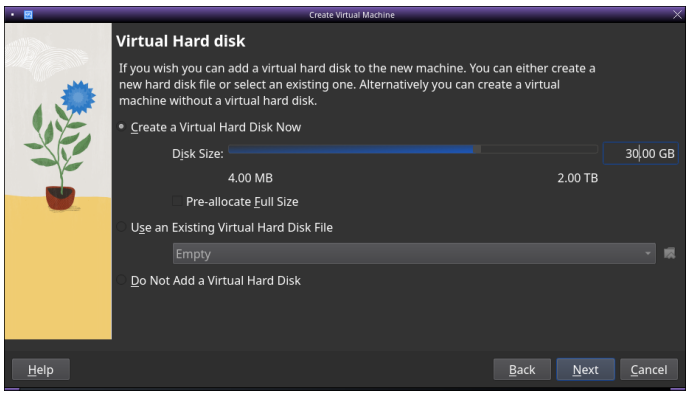

Figure 3.6: VirtualBox disk size

Note If you are installing something other than FreeBSD or GhostBSD in your VM, you'll need to manage virtual disk space carefully. Installing fasda-utils or multiqc via dreckly (or any other package manager that builds from source) will require the rust compiler, which needs about 14 GB of temporary space to build. If you use a VM with only a 30 GB virtual disk, install rust by itself first, then run **auto-dreckly-clean** to free all the temporary space used by the build, and then install fasda-utils and/or multiqc. Allocating a larger virtual disk (e.g. 50 GB) will make this unnecesssary. You should still run **auto-dreckly-clean** after installing packages to free up as much disk space as possible.

Now click on "Display" to adjust the display settings. GhostBSD uses the VirtualBox guest additions to allow resizing the virtual screen by simply dragging a corner of the window, support smooth mouse integration, etc. The guest additions work best with the VBoxSVGA virtual display device. Adjust as shown in Figure 3.7 and click "OK".

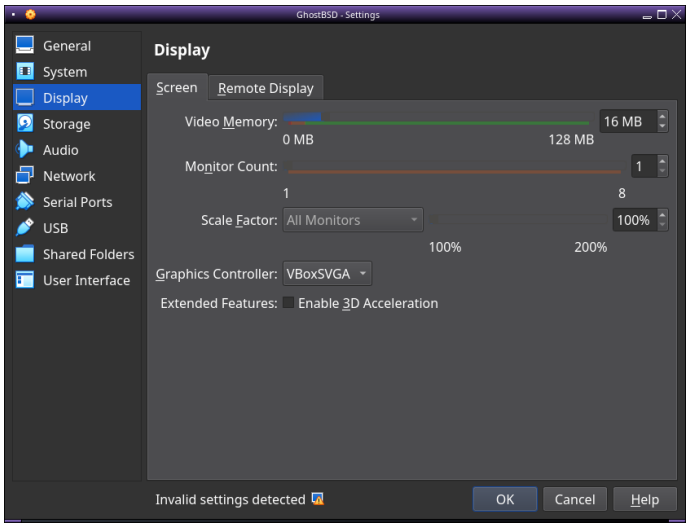

Figure 3.7: VirtualBox video emulation

Now we need to "insert" the virtual DVD. Click on "Storage", then click on "Empty" next to the optical disc icon under "Storage Devices", then on the optical disk icon in the upper right corner with a small arrow in it, as shown in Figure 3.8.

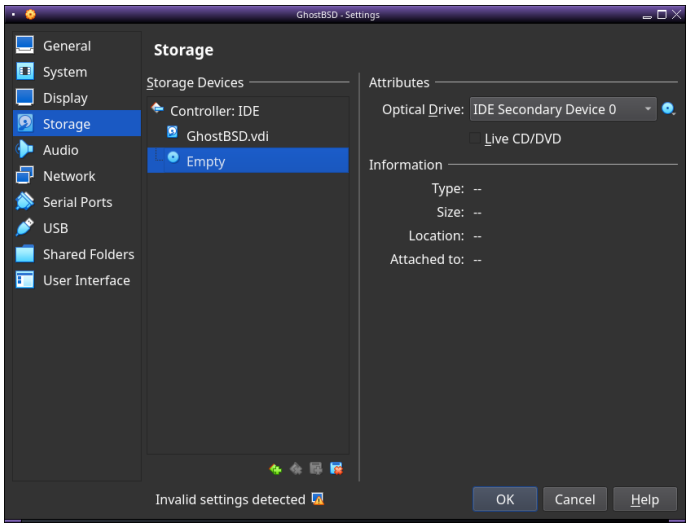

Figure 3.8: VirtualBox empty optical drive

This should open a menu as shown in Figure 3.9.

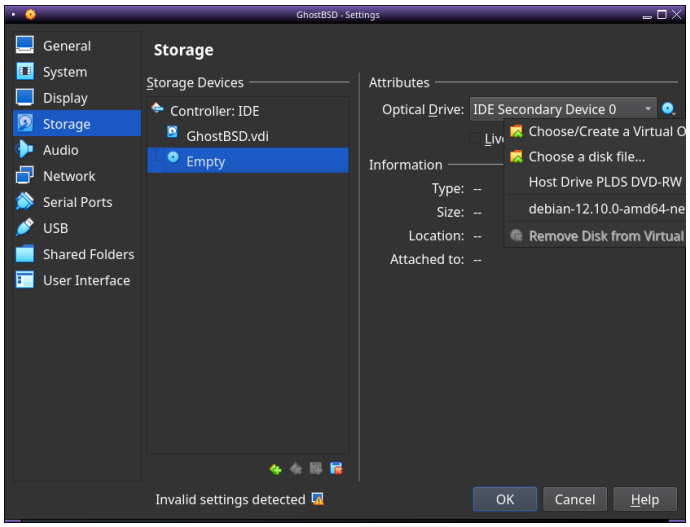

Figure 3.9: VirtualBox optical disk selection

Select "Choose a disk file" and load the GhostBSD ".iso" file you downloaded from the GhostBSD site, and click "Open". Figure 3.10 shows what this should look like.

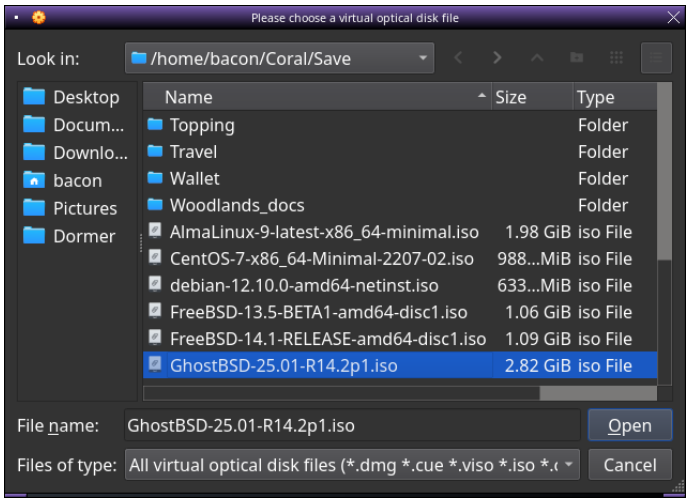

Figure 3.10: VirtualBox install image

After loading the virtual DVD (ISO file), the screen should look like Figure 3.11.

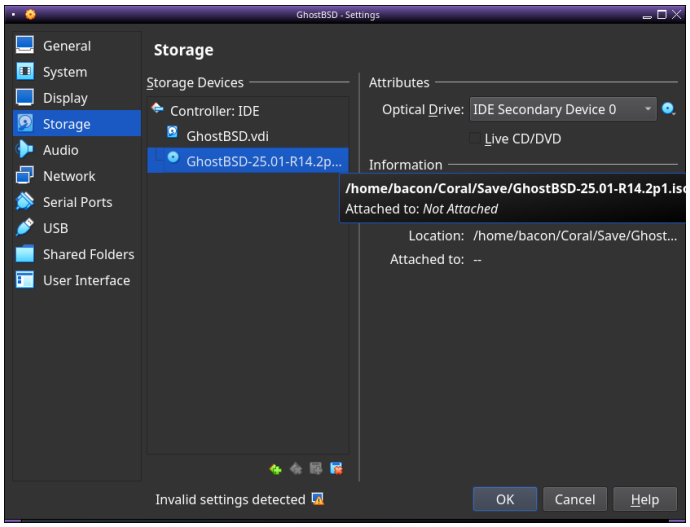

Figure 3.11: GhostBSD ISO file loaded

Now click "System" and adjust the boot order, as shown in Figure 3.12. Uncheck "Floppy" if it is checked, and use the arrows to the right to move devices up or down, until "Hard disk" is at the top, and "Optical" is second. This will cause the VM to boot from the ISO image first (because there is nothing on the hard disk), and then automatically boot from the hard disk after installation. You may also need to set "Enable hardware clock in UTC time", as shown. This will ensure that the VM guest shows the same time as the host, if the hardware clock uses UTC (Greenwich mean time).

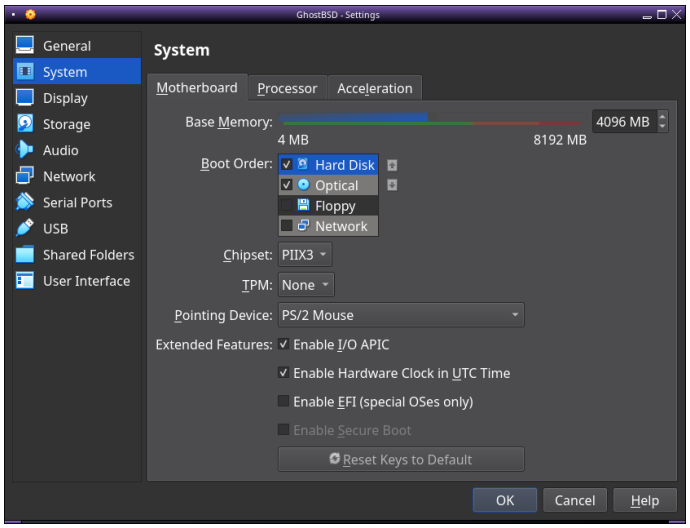

Figure 3.12: VirtualBox boot order

3.4.3 Installing GhostBSD

From here on, the process is the same whether you are installing in a virtual machine, or on real hardware (a.k.a. bare metal). Figure 3.13 shows the screen you should see after booting the VM from the ISO image, or the PC from a USB stick. Double-click on the "Install GhostBSD" icon on the desktop to get started.

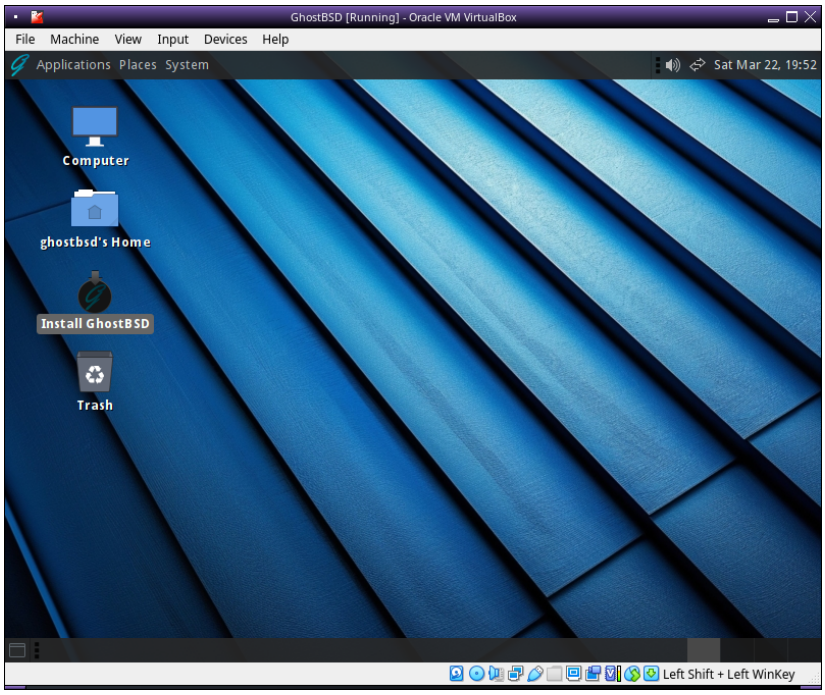

Figure 3.13: GhostBSD boot screen

You can just accept default responses to most questions, until you get to the account creation screen. Enter the information about the administrative user here, as shown in Figure 3.14. This could be an account like "Administrator", or your own name and username.

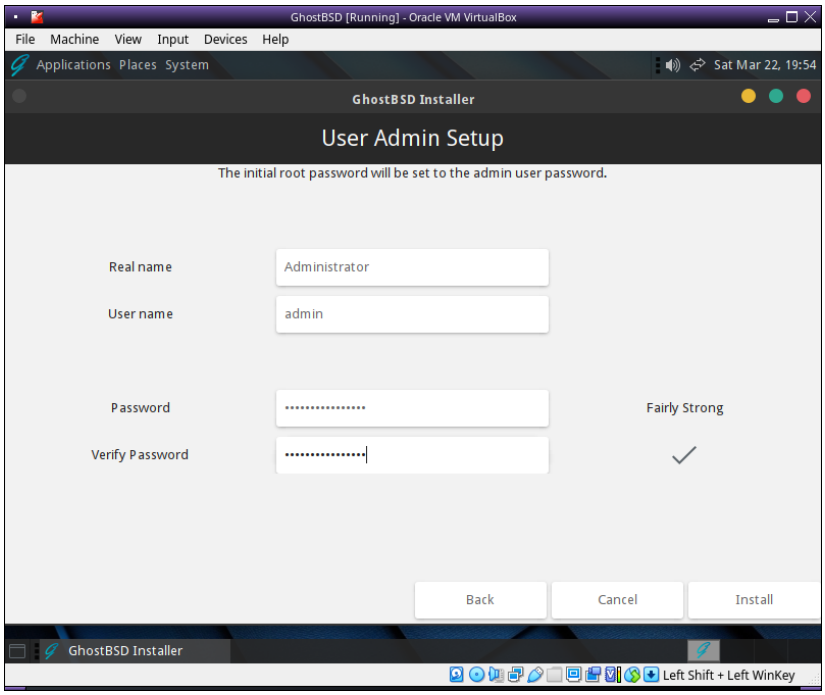

Figure 3.14: GhostBSD account creation

After the GhostBSD installation completes, the system will boot from the hard disk, assuming you changed the boot order when

setting up the VM. Otherwise, you will need to shut the VM down (use the Close/ACPI Shutdown option under the Machine menu), virtually eject the ISO image under "Storage", and boot again.

3.4.4 Getting started in GhostBSD

After booting from the hard disk and logging in, you can install the "rna-seq" meta-package from the "biology" category in Software Station, GhostBSD's graphical interface to the FreeBSD ports collection, as shown in Figure 3.15 and Figure 3.16.

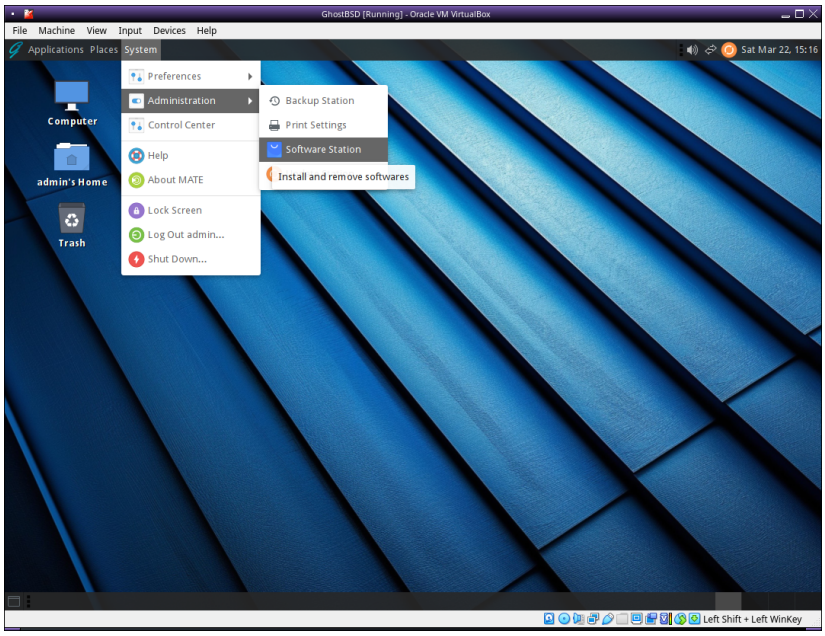

Figure 3.15: Running software station

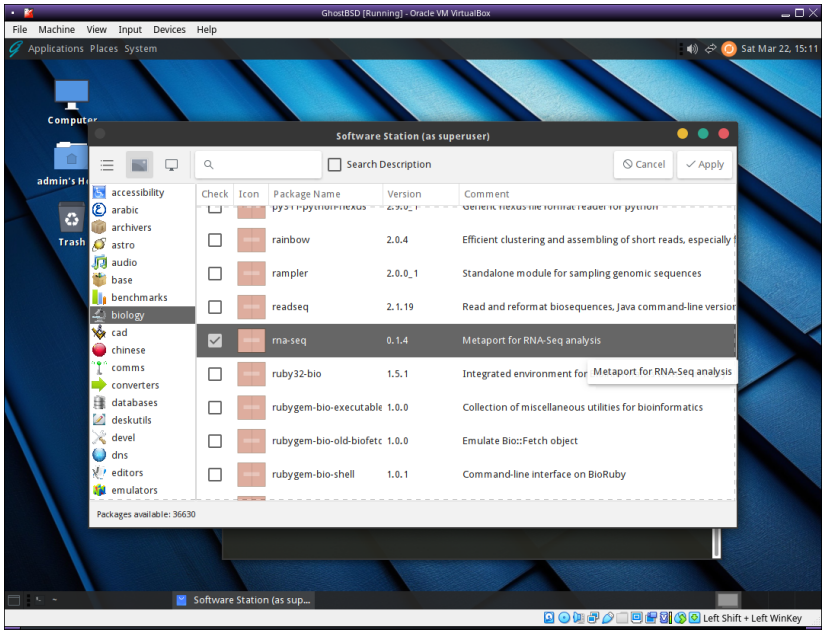

Figure 3.16: Selecting the rna-seq package

You'll want to install **git** via Software Station for downloading the scripts, etc. used for the example analysis in Chapter 4. Figure 3.17 shows how to do this using the Software Station search feature. You can also find it by browsing the "devel" category.

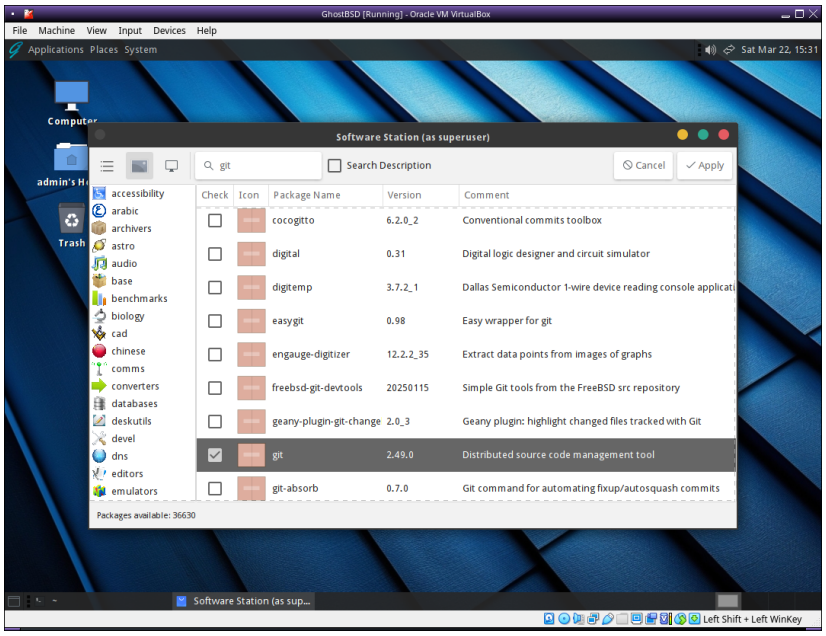

Figure 3.17: Installing git

To open a terminal emulator, follow the example in Figure 3.18. You should see a terminal window as shown in Figure 3.19.

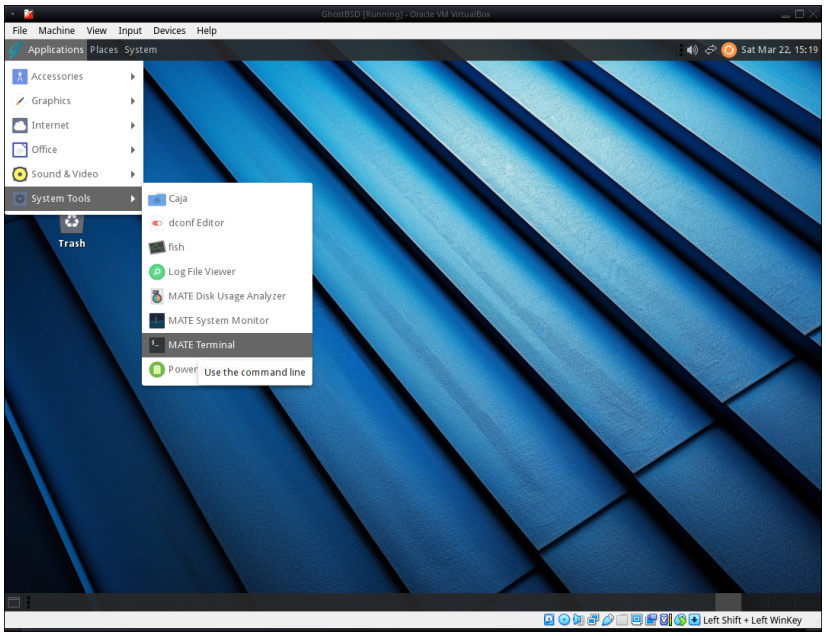

Figure 3.18: Opening a terminal emulator

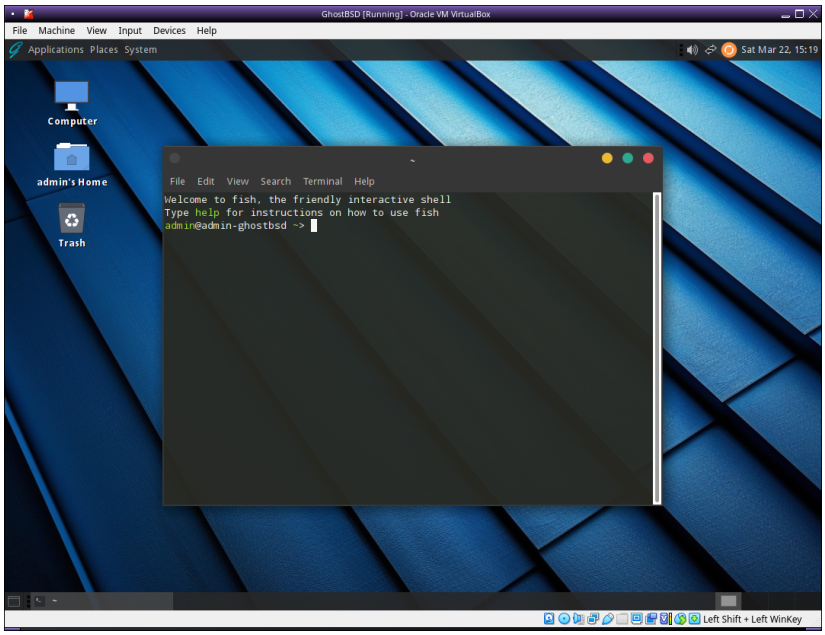

Figure 3.19: Terminal opened

At this point, you're ready to begin your RNA-Seq analysis!

3.4.5 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

---

1. Who can run GhostBSD in a virtual machine?
2. Briefly describe four advantages of GhostBSD for the RNA-Seq analysis in this text.
3. What are three major advantages of VirtualBox over other VMMs?
4. How is the process of installing GhostBSD different under VirtualBox, as opposed to installing on real hardware?
5. What is Software Station?

Chapter 4

RNA-Seq: A detailed example

4.1 A much easier approach

As mentioned earlier, some steps in a differential analysis have traditionally required advanced I.T. or computer programming skills in order to deploy or use the software. For example, the software tools necessary for an RNA-Seq analysis use a variety of different installation methods, some of which are rather elaborate. Computing fold-changes and associated P-values using most existing tools requires significant R programming skills, including knowledge of R data structures such as data frames and matrices. The time and effort required to learn R programming is a worthwhile investment for scientists who regularly perform complex statistical analyses. For others, it is a barrier that will delay or prevent completion of important work.

This text presents software packages that make software installation trivial, and new software tools that reduce the required knowledge to basic Unix command-line and scripting skills. These are the minimum skills required for most types of bioinformatics work anyway. Anyone hoping to do bioinformatics work in the future should first learn the Unix command-line and shell scripting. This is easily achieved in one semester. Other computer skills will be useful if your career involves frequent bioinformatics analyses, but they are not required for the RNA-Seq analysis presented here.

To eliminate the need for programming in the RNA-Seq differential expression analysis, we developed simple new command-line tools to replace those requiring the user to know R programming or manipulate data files between pipeline stages. Our tools cleanly interface with adjacent stages of the differential analysis pipeline that have traditionally required custom programming to manipulate the data files. New tools developed for this pipeline (and other purposes) include the following:

- Biolibc (https://github.com/orgs/auerlab/biolibc), a low-level C library supporting common bioinformatics file formats and algorithms. As it is written entirely in C and highly optimized for CPU and memory use, it provides maximum performance, and can be used from *any* popular programming language. All popular languages, including C++, Fortran, Python, Perl, and R, have facilities for utilizing C libraries.

  Biolibc-tools (https://github.com/orgs/auerlab/biolibc-tools) is a collection of simple tools based on biolibc, which are not complex enough to warrant their own project. It includes tools that provide summary statistics on FASTA/FASTQ files, compute chromosome lengths, convert Roman chromosome numbers to Arabic, etc. Think of them as the putty that fills in the small cracks in bioinformatics analyses, or the Swiss army knife that's good to have in your pocket for the occasions that you need it.

  FastQ-Trim (https://github.com/orgs/auerlab/fastq-trim) is a near-optimal read trimmer, designed as a drop-in replacement for tools such as **cutadapt** and **Trimmomatic**, but significantly faster.

  FASDA (https://github.com/orgs/auerlab/fasda) is a 100% C differential analysis tool, replacing existing tools that require the user to know R programming and data structures. It is trivial to install via a package manager, and easy to use, directly reading SAM/BAM/CRAM output from traditional aligners, or abundances from **kallisto** output. It includes tools for filtering output and generating heatmaps using simple commands.

  Peak-classifier (https://github.com/orgs/auerlab/peak-classifier) is a simple command-line tool for quickly classifying ATAC-Seq or ChIP-Seq peaks as intronic, intergenic, etc. It includes a Python matplotlib script for viewing features graphically.

  Basic-stats (https://github.com/orgs/auerlab/basic-stats) is a command-line tool for performing basic statistical analyses on tabular input files. It can compute means, medians, variance, etc. on any row or column of an input file. Unlike R scripts,

which typically inhale all data into memory before processing it, basic-stats uses a trivial amount of memory regardless of the size of the input file. Hence, huge amounts of data can be processed on any computer, even a low-end laptop.

Microsynteny-tools (https://github.com/orgs/auerlab/microsynteny-tools) is a suite of tools for comparing gene neighborhoods across species. This aids in understanding evolutionary changes that may have occurred between fish, amphibians, and mammals, for example.

All of these tools are open source and available in the FreeBSD ports and dreckly package managers, so that they are trivial to install on any Unix-compatible system. Some of them have also been ported to other package managers, such as AUR.

4.1.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

---

1. What is the major hurdle for biologists who want to compute fold-changes and P-values using most existing software?
2. What is a hurdle for doing scientific computing in general?
3. What are the minimum skills needed to do most kinds of bioinformatics?

4.2 A programming-free work flow

As stated earlier, some steps in the RNA-Seq analysis process are fairly straightforward, while others are traditionally performed using hard-to-use tools. Additionally, there have been no well-established tools available for many of the smaller tasks, such as computing chromosome sizes for Kallisto, dealing with Roman numerals in genomic files, converting Ensembl gene IDs to canonical gene names, converting between GFF and BED formats, etc. Researchers often end up writing their own Python or perl scripts for tasks like these, or using cumbersome online tools that disrupt the workflow. This is a waste of time individually, and a lot of duplicated effort overall. Having a well-written, generalized command-line tool for tasks like these makes many future analyses easier for everyone.

Figure 4.1 depicts a simplified flowchart for our RNA-Seq analysis pipeline using our new tools in place of traditional tools. Details of this workflow are described in the sections that follow.

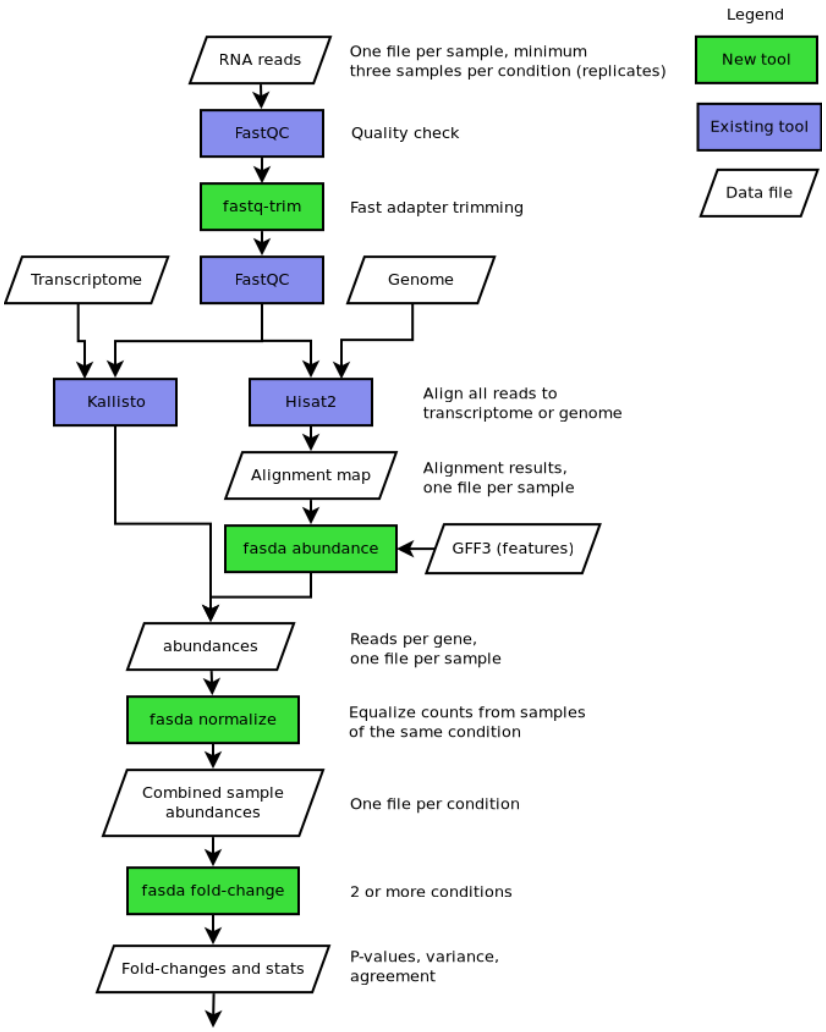

Figure 4.1: FASDA RNA-Seq workflow

Existing tools used in this workflow are selected mainly on the basis of efficiency, reliability, and ease of use. The reader is encouraged to try alternative tools for every stage of every analysis. Firsthand experience is the only truly reliable data. Relying on (possibly dated) literature or the advice of others will generally lead to suboptimal results.

4.2.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. How strictly should you adhere to the tools listed in the flowchart of this section?

4.3 Saving terminal output

One of the great features of Unix is *device independence*, which makes input/output (I/O) to/from any device (e.g. keyboard, mouse, printer, scanner, etc.) work the same way from any program. In fact, I/O to/from a device works exactly like I/O to/from an ordinary file.

There are two different output streams that every Unix process (running program) has open that normally go to the terminal screen (window):

- The *standard output* (often abbreviated *stdout*) is for "normal" output, i.e. results from the computation or general information for the user.
- The *standard error* (abbreviated stderr) is mainly for warning and error messages.

Similarly, the *standard input* (stdin) is a stream that every Unix process is born with, and normally comes from the terminal keyboard.

Many of the commands we run here will produce a large amount of output to the terminal screen. You may want to save this output to a file so you can review it using **more**, search it using **grep**, etc. This can be easily done for any command using the Unix using *redirection* which disconnects the standard streams from the terminal and connects them to a file or device of our choosing. For example, to save the output of the **ls** command to a file, we could do the following:

```
shell-prompt: ls -l > file-listing.txt
```

Similarly, *pipes* disconnect one of the standard streams from the terminal and connect it to a standard stream of another Unix process. For example, to list files and view the output one page at a time, we could pipe the **ls** output through **more**:

```
shell-prompt: ls -l | more
```

These Unix features are covered in detail in The Research Computing User's Guide.

To send a copy of terminal output to a file *and* still see it on the screen, we can use the Unix **tee** command, which takes input from stdin, duplicates it, and sends the two copies to two different streams, namely stdout and the filename provided as an argument to **tee**. The name **tee** comes from its resemblance to a tee in your plumbing that allows water to flow from a single source to multiple faucets. For example, output from the script in the next section can be saved as follows:

```
# Redirect stdout from 01-fetch.sh to tee, and send a copy to the
# file fetch.stdout
shell-prompt: ./01-fetch.sh 3 | tee fetch.stdout
```

In some Unix shells, such as **bash** and **tcsh**, we can pipe both stdout and stderr using the 'l&' operator:

```
# Redirect stdout and stderr from to tee, and send a copy to fetch.out
shell-prompt: ./01-fetch.sh 3 |& tee fetch.out
```

In many Bourne-shell compatible shells, we need to pipe or redirect the streams separately, using a rather Unix-geeky syntax utilizing the integer file descriptors 2 (for stderr) and 1 (for stdout). In this case, stderr must be redirected first, and stdout piped afterward. This syntax is a bit cryptic for non Unix gurus, but the advantage is that it allows us to redirect stdout and stderr to different files if we so desire. For our purposes here, we'll just save them both the same file, which will then contain an exact copy of what we see on the terminal.

```
# Redirect stdout and stderr from to tee, and send a copy to fetch.out
shell-prompt: ./01-fetch.sh 3 2>&1 | tee fetch.out
```

4.3.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. What is device independence?
2. What are the three standard streams open by default in every Unix process, and to what file or device are they normally connected?
3. What is redirection?
4. What are pipes?
5. Show how to list the files in the current directory, save the output to "listing.txt", and see it on the screen at the same time.

4.4 Obtaining raw reads

If you're doing an RNA-Seq differential analysis, odds are you sent your RNA library to a sequencing center, and downloaded the FASTQ files following their instructions. For the sake of instruction, we will use preexisting data from a study comparing wild-type and mutant strains of the common yeast, saccharomyces cerevisiae [Schurch]. These data have a number of advantages for studying RNA-Seq differential analysis:

- The yeast genome is quite small, so analysis steps can be performed quickly, even on an low-end PC or virtual machine.
- Saccharomyces cerevisiae is a common model organism, so the genome and transcriptome are fairly complete.
- This data set includes 48 biological replicates. We can compare the results of a typical three-replicate analysis to those using many more replicates, and compare results from different sets of three replicates to see how widely they vary.
- The data set includes seven technical replicates (divisions of the RNA sample from the same biological sample). This allows us to see how much variation can result from technical factors (factors other than biological sample variation).

These data are available from the *Sequence Read Archive* (SRA), at https://www.ncbi.nlm.nih.gov/bioproject/PRJEB5348. The SRA is a global repository containing the vast majority of all sequence data used in prior published studies. We will use the sra-tools command-line suite to download the raw reads. This software suite facilitates fast downloading of SRA data, including filtering on the SRA server end to avoid costly downloads of data we don't need. is installed by the rna-seq meta-package in both FreeBSD ports and dreckly packages, that you should have already installed following the instructions in Chapter 3.

A script is provided for automatically fetching the raw data files you need from the SRA. It is not critical for you to understand the script at this moment, but it's a good example to study to help you script future analyses. This script, and other scripts for this RNA-Seq pipeline, are available at https://github.com/outpaddling/DIY-RNA-Seq-DE. Simply clone the repository as shown below, then **cd** into the directory, and run the first script, **01-fetch.sh**, followed by the number of biological replicates to use in the analysis. Most RNA-Seq experiments, unfortunately, use only three replicates due to cost constraints. This will not yield reliable results, but this is fine for learning the mechanics of an analysis, and it will make the analysis stages go very quickly.

As mentioned earlier, "shell-prompt: " is used throughout this text to represent the Unix shell prompt, which will be different in your terminal emulator windows. In any case, it is an indication from the Unix shell that it is waiting for you to enter another command. Type the text following "shell-prompt: " in all of the examples presented here. Any text following the character '#' is a comment to help explain the command, and is not to be typed in.

Note If you see an arrow at the end of a line in any text block like the one below, it means that the line was tool long for the page, and is continued on the next line.

```
shell-prompt: git clone https://github.com/outpaddling/DIY-RNA-Seq-DE.git
shell-prompt: cd DIY-RNA-Seq-DE/Yeast
shell-prompt: ./01-fetch.sh 3

Darwin tarpon.acadix.biz 24.3.0 Darwin Kernel Version 24.3.0: Thu Jan  2 20:24:06 PST 2025; ↩
    root:xnu-11215.81.4~3/RELEASE_ARM64_T8103 arm64

fasterq-dump : 3.2.1

/Users/bacon/DIY-RNA-Seq-DE/Yeast
WT:
Downloading ERR458493...
2025-03-29T12:44:43 prefetch.3.2.1: 1) Resolving 'ERR458493'...
2025-03-29T12:44:45 prefetch.3.2.1: Current preference is set to retrieve SRA Normalized ↩
   Format files with full base quality scores
2025-03-29T12:44:45 prefetch.3.2.1: 1) Downloading 'ERR458493'...
2025-03-29T12:44:45 prefetch.3.2.1:  SRA Normalized Format file is being retrieved
2025-03-29T12:44:45 prefetch.3.2.1:  Downloading via HTTPS...
|---------------------------------------------------- 100%
```

```
2025-03-29T12:44:47 prefetch.3.2.1:  HTTPS download succeed
2025-03-29T12:44:48 prefetch.3.2.1:  'ERR458493' is valid: 41937532 bytes were streamed  ↩
    from 41923131
2025-03-29T12:44:48 prefetch.3.2.1: 1) 'ERR458493' was downloaded successfully
join   :|-------------------------------------------------- 100%
concat :|-------------------------------------------------- 100%
spots read      : 1,093,957
reads read      : 1,093,957
reads written   : 1,093,957
Compressing...

[ Output omitted for brevity ]

Downloading ERR458514...
Read:     0   B /   370 MiB ==>  0%2025-03-29T12:45:19 prefetch.3.2.1: 1) Resolving ' ↩
    ERR458514'...
Results/01-fetch/ERR458507.fastq : 25.07%   (   370 MiB =>   92.8 MiB, Results/01-fetch/ ↩
    ERR458507.fastq.zst)
2025-03-29T12:45:20 prefetch.3.2.1: Current preference is set to retrieve SRA Normalized  ↩
    Format files with full base quality scores
2025-03-29T12:45:21 prefetch.3.2.1: 1) Downloading 'ERR458514'...
2025-03-29T12:45:21 prefetch.3.2.1:  SRA Normalized Format file is being retrieved
2025-03-29T12:45:21 prefetch.3.2.1:  Downloading via HTTPS...
|-------------------------------------------------- 100%
2025-03-29T12:45:24 prefetch.3.2.1:  HTTPS download succeed
2025-03-29T12:45:24 prefetch.3.2.1:  'ERR458514' is valid: 60849902 bytes were streamed  ↩
    from 60842523
2025-03-29T12:45:24 prefetch.3.2.1: 1) 'ERR458514' was downloaded successfully
join   :|-------------------------------------------------- 100%
concat :|-------------------------------------------------- 100%
spots read      : 1,594,695
reads read      : 1,594,695
reads written   : 1,594,695
Compressing...
Results/01-fetch/ERR458514.fastq : 25.07%   (   370 MiB =>   92.9 MiB, Results/01-fetch/ ↩
    ERR458514.fastq.zst)
total 1125216

-rw-r--r--  1 bacon  staff   66840125 Mar 29 07:44 ERR458493.fastq.zst
-rw-r--r--  1 bacon  staff  115305209 Mar 29 07:45 ERR458500.fastq.zst
-rw-r--r--  1 bacon  staff   97338248 Mar 29 07:45 ERR458507.fastq.zst
-rw-r--r--  1 bacon  staff   97377341 Mar 29 07:45 ERR458514.fastq.zst
-rw-r--r--  1 bacon  staff   89866345 Mar 29 07:44 ERR458878.fastq.zst
-rw-r--r--  1 bacon  staff   75157119 Mar 29 07:45 ERR458885.fastq.zst
```

```
# To save terminal output to a file:
shell-prompt: ./01-fetch.sh 3 |& tee fetch.out

# If the syntax above doesn't work in your shell:
shell-prompt: ./01-fetch.sh 3 2>&1 | tee fetch.out
```

Note

Be sure to carefully review the output to the terminal for warnings, errors, and reasonable values. If you saved a copy of the terminal output by piping through **tee**, you can use tools such as **grep -i error file.out** to search for error messages, etc.

4.4.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. What makes saccharomyces cerevisiae a good organism for practicing RNA-Seq analysis?
2. What is the SRA?
3. What is sra-tools?

4.5 Recompression

Sequence files are often compressed using **gzip**, one of the oldest compression tools in use today. More modern tools offer a much better *compression ratio*, the ratio of the uncompressed / compressed file size.

The **xz** command typically produces compressed FASTQ files around 40% smaller than **gzip**. Compression with **xz** is much slower than **gzip**, but decompression with **unxz** is about as fast as **gunzip**.

Hence, **xz** is a good choice for files that will be stored long-term (archived). E.g., if you have 200 GB of raw sequence data in **gzip** format, you can likely reduce it to around 120 GB by recompressing with **xz**. An extra 80 GB of free disk space could make the difference between completing an analysis and having it stalled due to a full disk, or having to suffer the expense and hassle upgrading your storage. This is especially important when performing analyses on a typical PC or Mac, where storage may be limited. Recompressing data is simple:

```
# Recompress gzipped data with xz
# Note: Some systems have a "zcat" command, which is equivalent to
# "gunzip -c", while others use "gzcat".  We always use "gunzip -c"
# to avoid portability problems in our commands and scripts.
shell-prompt: gunzip -c file.fastq.gz | xz > file.fastq.xz

# Verify: The following two commands should produce the same output.
# "md5" computes a "fingerprint" for a file, which is astronomically
# unlikely to be the same for two different files.  Hence, if the two
# commands below produce the same fingerprint, we can be confident that
# the files are identical.
# We could also use "xzcat" in place of "unxz -c", but "gunzip -c"
# would be "zcat" on some systems and "gzcat" on others.  We always use
# "gunzip -c" to avoid annoying portability issues.
shell-prompt: gunzip -c file.fastq.gz | md5
shell-prompt: unxz -c file.fastq.xz | md5

# If verified, remove the original gzipped file to reclaim disk space
shell-prompt: rm file.fastq.gz
```

Note If you want to know more about and of the commands you see in these examples, just type **man command-name**, e.g. **man gunzip**, **man unxz**, or **man rm**.

For temporary files generated during analysis, **xz** would likely just slow things down, as **xz** compression may actually take longer than common tasks like read trimming. Other tools such as **zstd** or **lz4** are better options, since they are much faster. The **zstd** command has effectively obsoleted **gzip**, since it is much faster and typically provides equal or better compression. The **lz4** command is even faster, but does not compress nearly as well. The data below show the size of a raw FASTQ file alongside the same file compressed with common tools. Another common compression tool is **bzip2**, which offers a better compression ratio than **gzip**, but is slower for both compression and decompression, and does not approach the compression ratio of **xz**. Hence, **bzip2** is no longer popular since the advent of newer tools.

```
shell-prompt: ls -l sample16*
-rw-r-----  1 bacon  bacon    19G Jun 21 10:33 sample16-time000-rep1-R1.fastq
-rw-r-----  1 bacon  bacon   6.3G Jun 21 11:26 sample16-time000-rep1-R1.fastq.lz4
-rw-r-----  1 bacon  bacon   3.9G Jun 20 20:46 sample16-time000-rep1-R1.fastq.gz
-rw-r-----  1 bacon  bacon   3.7G Jun 21 11:23 sample16-time000-rep1-R1.fastq.zstd
-rw-r-----  1 bacon  bacon   2.3G Jun 21 11:23 sample16-time000-rep1-R1.fastq.xz
```

Note
The ZFS filesystem uses lz4 compression for all files by default on most systems that support it. Hence, if you store `sample16-time000-rep1-R1.fastq` on a ZFS filesystem, **ls -l** will show it as 19G as above, but it will only use about 6.3GB of disk space in reality. The disk usage command **du -h sample16-time000-rep1-R1.fastq** will reveal this difference. ZFS is used unconditionally by GhostBSD and SunOS, and is one of two options offered by the FreeBSD installer. Some GNU/Linux systems also support booting from ZFS, while others only support creating ZFS filesystems after installation.
Even if you use ZFS with compression, it is still worthwhile compressing your files in most cases, especially if they reside on a remove file server. For example, when reading the raw file `sample16-time000-rep1-R1.fastq` above from a remote file server using ZFS, the file server would decompress it and transfer 19GB across the network to your PC. If reading `sample16-time000-rep1-R1.fastq.zst` from a remote file server, it would only transfer 3.7GB over the network, and decompression would occur on your PC. The same savings would occur when writing compressed files vs relying on the server's ZFS compression.

At the time of this writing, many bioinformatics tools offer spotty support for compression tools. For example, they may directly support reading and writing files compressed with **gzip**, but not **zstd** or **xz**. This is usually easy to work around, but if you want to keep things as simple as possible for your first RNA-Seq analysis, you may choose to just use **gzip** even though it is inferior to more modern tools.

All of the new tools we developed (FASDA, fastq-trim, etc.) innately support all common compression tools, owing to their use of the `xt_fopen()` function from libxtend.

All of the common compression tools allow the user to control the compression level in order to trade between performance and compression ratio. To prevent intermediate file compression from becoming a bottleneck for your analysis, just use a lower compression level. This will cause the compressed files to be slightly bigger in exchange for faster processing. For example, the **gzip** command offers nine levels from 1 (fastest and minimum compression) to 9 (slowest and maximum compression). The default is 6. This doesn't matter much unless the analysis tool might be faster than the compression tool. Read trimming (discussed in the sections that follow) often takes less time than compressing the output files, so you might speed it up with careful selection of compression tools and options. Mapping reads to a reference genome (also discussed later) will generally take much longer than compressing the resulting output, so don't bother worrying about a compression bottleneck here.

```
# Default compression of 6
shell-prompt: my-analysis-tool input1.fastq | gzip > output1.gz

# Lower compression level, resulting in slightly larger output1.gz,
# but faster processing
shell-prompt: my-analysis-tool input1.fastq | gzip -1 > output1.gz
```

4.5.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. What is the best compression tool for sequence data that will be stored long-term?
2. What is the best compression tool for temporary files containing sequence data?
3. Some filesystems, such as ZFS, can automatically compress all of our files. Does this render compression tools useless?

4. If a compression tool is slowing down an analysis by using too much CPU time, does this mean we should switch to a different tool?

5. How can we easily write a program that can read or write files compressed with any of the common tools?

4.6 Descriptive naming to avoid calamities

One of the important lessons learned from our early RNA-Seq / ATAC-Seq analyses was the need for clear and descriptive naming of the raw data files. Before developing an analysis pipeline, a little preparation can make everything that follows much simpler and less confusing. The filenames of the raw FASTQ files provided by the sequencing center often have extremely cryptic names that are both difficult for humans to decipher and difficult for analysis scripts to parse (dissect). They may not even be consistent. For example, for one of our projects, we received FASTQ files like those below. The leftmost "word" in each line ending in ".fq", is the filename, and everything after is our description of the file.

```
# Key for interpreting filenames
# A = axolotl
# XX (e.g. 16) = sample number
# Na, 26h, d1 = time points
# _1 = forward reads, _2 = reverse reads (paired-end sequencing)

A16Na1_1.fq  Sample 16, naive (time 0), replicate 1, forward reads
A16Na1_2.fq  Sample 16, naive (time 0), replicate 1, reverse reads
A17Na2_1.fq  Sample 17, naive (time 0), replicate 2, forward reads
A1926h1_1.fq Sample 19, 26 hours post-fertilization, replicate 1, forward reads
A2026h2_1.fq Sample 20, 26 hours post-fertilization, replicate 2, forward reads
A285d1_1.fq  Sample 28, 5 days post-fertilization, replicate 1, forward reads
A295d2_1.fq  Sample 29, 5 days post-fertilization, replicate 2, forward reads
```

Note how the filenames are impossible to interpret without being given instructions, and that the time points do not even use consistent nomenclature (Na = naive, XXh = hours, dX = days). Dealing with filenames like these throughout an analysis will make the scripts more complicated and confusing, and will likely lead to mistakes. In this case, the researcher requested more descriptive names, but was told by the sequencing center that they were limited to nine characters. I haven't run into such a filename limitation since using MS-DOS in the 1980s, but sometimes it's not worth arguing, and we just need to work around the problem.

For another analysis we performed, consisting of RNA-Seq and ATAC-Seq data from mouse neural crest cells, a similarly cryptic naming system was used, where letters A, B, and C represented time points and digits 1, 2, and 3 represented replicates. Worse yet, the digits and letters were mistakenly transposed for the ATAC-Seq data. This led to many wasted hours reviewing the analysis process and lab documentation in order to determine why we were seeing very few significant changes in chromatin accessibility across time points. It turned out that we were comparing replicates from the same time point to each other, rather than comparing different time points as intended.

Ideally, more descriptive names should be assigned by the sequencing center, so there is no source of confusion to begin with. If this does not happen, then the first step in our analysis is to decrypt the filenames. On all Unix systems, this is easy to do, without altering the original filenames, using *links*, which simply create another filename for the same file. This is demonstrated below.

Using three replicates of the yeast data, we obtain the following six raw data files from the SRA:

```
ERR458493.fastq.zst  ERR458507.fastq.zst  ERR458878.fastq.zst
ERR458500.fastq.zst  ERR458514.fastq.zst  ERR458885.fastq.zst
```

The filename stems here are SRA *accession numbers*, which in no way describe the experimental parameters of the samples. We will need information from the study to decode them and determine sample number, condition, and replicate. For these data, a TSV (tab-separated-values) file from the original study is provided in the DIY-RNA-Seq-DE repository to help map the SRA accession numbers to replicate and condition. We can look at the first 10 lines of the file using the Unix **head** command:

```
shell-prompt: head ERP004763_sample_mapping.tsv
RunAccession    Lane    Sample  BiolRep
ERR458493       1       WT      1
ERR458494       2       WT      1
```

```
ERR458495       3       WT      1
ERR458496       4       WT      1
ERR458497       5       WT      1
ERR458498       6       WT      1
ERR458499       7       WT      1
ERR458500       1       SNF2    1
ERR458501       2       SNF2    1
```

"Lane" in this file refers the the physical location of the sample on the sequencer slide, and is equivalent to the technical replicate (a partition of the same biological sample). Using this information, we can find out about our raw files, and manually create the links:

```
# Get the accession numbers
shell-prompt: ls Results/01-fetch
ERR458493.fastq.zst  ERR458507.fastq.zst  ERR458878.fastq.zst
ERR458500.fastq.zst  ERR458514.fastq.zst  ERR458885.fastq.zst

# grep is a Unix command to show lines in a file containing some pattern
# Show lines in our map file containing any of the accession numbers
shell-prompt: grep -E 'ERR458493|ERR458507|ERR458878|ERR458500|ERR458514|ERR458885'  ↩
    ERP004763_sample_mapping.tsv
ERR458493       1       WT      1
ERR458500       1       SNF2    1
ERR458507       1       SNF2    2
ERR458514       1       SNF2    3
ERR458878       1       WT      2
ERR458885       1       WT      3

# Manually create links with meaningful names
shell-prompt: mkdir Results/02-readable-names
shell-prompt: cd Results/02-readable-names
shell-prompt: ln ../01-fetch/ERR458493.fastq.zst WT-01.fastq.zst
shell-prompt: ln ../01-fetch/ERR458493.fastq.zst sample01-cond1-rep01.fastq.zst

Etcetera...

# Back to the Yeast directory
shell-prompt: cd ../..
```

Rather than make you muddle around with this, a script is provided in the DIY-RNA-Seq-DE repository which will create the links automatically. It creates links with generic "condition X replicate Y" names, as well as links with descriptive names "WT" for wild type (condition 1) and "SNF2" for the mutant strain. This way, we can simply reference these links if we ever forget what is what. It is not critical for you to understand the script at this moment, but it's a good example to study to help you script future analyses.

```
shell-prompt: ./02-readable-names.sh

Darwin tarpon.acadix.biz 24.3.0 Darwin Kernel Version 24.3.0: Thu Jan  2 20:24:06 PST 2025; ↩
     root:xnu-11215.81.4~3/RELEASE_ARM64_T8103 arm64
/Users/bacon/DIY-RNA-Seq-DE/Yeast
Linking ERR458493 = WT-1 = cond1-rep1...
Linking ERR458500 = SNF2-1 = cond2-rep1...
Linking ERR458507 = SNF2-2 = cond2-rep2...
Linking ERR458514 = SNF2-3 = cond2-rep3...
Linking ERR458878 = WT-2 = cond1-rep2...
Linking ERR458885 = WT-3 = cond1-rep3...
total 0
lrwxr-xr-x  1 bacon  staff  31 Mar 29 07:46 SNF2-01.fastq.zst -> ../01-fetch/ERR458500. ↩
    fastq.zst
lrwxr-xr-x  1 bacon  staff  31 Mar 29 07:46 SNF2-02.fastq.zst -> ../01-fetch/ERR458507. ↩
    fastq.zst
```

```
lrwxr-xr-x  1 bacon  staff  31 Mar 29 07:46 SNF2-03.fastq.zst -> ../01-fetch/ERR458514. ↩
    fastq.zst
lrwxr-xr-x  1 bacon  staff  31 Mar 29 07:46 WT-01.fastq.zst -> ../01-fetch/ERR458493.fastq. ↩
    zst
lrwxr-xr-x  1 bacon  staff  31 Mar 29 07:46 WT-02.fastq.zst -> ../01-fetch/ERR458878.fastq. ↩
    zst
lrwxr-xr-x  1 bacon  staff  31 Mar 29 07:46 WT-03.fastq.zst -> ../01-fetch/ERR458885.fastq. ↩
    zst
lrwxr-xr-x  1 bacon  staff  31 Mar 29 07:46 sample01-cond1-rep01.fastq.zst -> ../01-fetch/ ↩
    ERR458493.fastq.zst
lrwxr-xr-x  1 bacon  staff  31 Mar 29 07:46 sample02-cond2-rep01.fastq.zst -> ../01-fetch/ ↩
    ERR458500.fastq.zst
lrwxr-xr-x  1 bacon  staff  31 Mar 29 07:46 sample03-cond2-rep02.fastq.zst -> ../01-fetch/ ↩
    ERR458507.fastq.zst
lrwxr-xr-x  1 bacon  staff  31 Mar 29 07:46 sample04-cond2-rep03.fastq.zst -> ../01-fetch/ ↩
    ERR458514.fastq.zst
lrwxr-xr-x  1 bacon  staff  31 Mar 29 07:46 sample05-cond1-rep02.fastq.zst -> ../01-fetch/ ↩
    ERR458878.fastq.zst
lrwxr-xr-x  1 bacon  staff  31 Mar 29 07:46 sample06-cond1-rep03.fastq.zst -> ../01-fetch/ ↩
    ERR458885.fastq.zst
```

```
# To save terminal output to a file:
shell-prompt: ./02-readable-names.sh |& tee readable-names.out

# If the syntax above doesn't work in your shell:
shell-prompt: ./02-readable-names.sh 2>&1 | tee readable-names.out
```

Note
Be sure to carefully review the output to the terminal for warnings, errors, and reasonable values. If you saved a copy of the terminal output by piping through **tee**, you can use tools such as **grep -i error file.out** to search for error messages, etc.

The links created above, such as sample16-cond1-rep1.fastq.gz, are both easy for people to understand and easy for scripts to dissect. Extracting the sample number, condition, or replicate is trivial compared to doing the same with the original filenames. Since the filenames are so generic, we can also reuse the scripts for other analyses with minimal changes.

4.6.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. Describe two reasons we should give our raw data files clear and descriptive names.
2. What can we do if the sequencing center gives us files with cryptic names, without losing any information about the original names, and without using additional disk space?

4.7 Pre-trim quality check

In line with the "know your data" principle, we start by simply looking at the raw data. Here we can see the PHRED-encoded quality scores. We can use the **phred-decode** script from Section 1.5 to determine the read scores, which are quite high. A copy of this script is included in the DIY-RNA-Seq-DE repository for your convenience.

```
# zstdcat uncompresses a file and sends output to the standard output,
# which is normally the terminal window.
# We pipe this output through "more" to show it one screen at a time.
# Type 'h' for help using more.
shell-prompt: zstdcat Results/02-readable-names/sample01-cond1-rep01.fastq.zst | more
@ERR458493.1 DHKW5DQ1:219:D0PT7ACXX:1:1101:1724:2080 length=51
CGCAAGACAAGGCCCAAACGAGAGATTGAGCCCAATCGGCAGTGTAGTGAA
+ERR458493.1 DHKW5DQ1:219:D0PT7ACXX:1:1101:1724:2080 length=51
B@@FFFFFHHHGHJJJJJJIJJGIGIIIGI9DGGIIIEIGIIFHHGGHJIB
@ERR458493.2 DHKW5DQ1:219:D0PT7ACXX:1:1101:2179:2231 length=51
ACTAATCATCAACAAAACAATGCAATTCAAGACCATCGTCGCTGCCTTCGC
+ERR458493.2 DHKW5DQ1:219:D0PT7ACXX:1:1101:2179:2231 length=51
B@=DDFFFHHHHHJJJJIJJJJJJIJJJJJJJJJJJJJJJJJJJIJJJI
@ERR458493.3 DHKW5DQ1:219:D0PT7ACXX:1:1101:2428:2116 length=51
CTCAAAACGCCTACTTGAAGGCTTCTGGTGCTTTCACCGGTGAAAACTCCG
+ERR458493.3 DHKW5DQ1:219:D0PT7ACXX:1:1101:2428:2116 length=51
@@@DDFFFHDHHFHJJGGIGIIIIJIJJFG@HIIIJFGGIHHHJIGEHIGC

[ Output omitted for brevity ]

# Decode first several PHRED scores from the first read.
# Put the string in hard (single) quotes, as it may contain characters
# that the Unix shell considers special.
shell-prompt: ./phred-decode 'B@@FFFFFHH'
Chr     ASCII   QUAL    P(error)
B       66      33      .00050118723362727228
@       64      31      .00079432823472428150
@       64      31      .00079432823472428150
F       70      37      .00019952623149688796
F       70      37      .00019952623149688796
F       70      37      .00019952623149688796
F       70      37      .00019952623149688796
F       70      37      .00019952623149688796
H       72      39      .00012589254117941672
H       72      39      .00012589254117941672
```

From the output above, we can see what our input files look like in general, and the spot-check of a few PHRED scores shows very high quality reads. This does not tell us anything about the quality of the data in general, though. We can get some summary information about the raw data files using **blt fastx-stats**, which is part of the biolibc-tools suite, which in turn is installed with the rna-seq meta-package.

```
shell-prompt: blt fastx-stats Results/02-readable-names/sample01-cond1-rep01.fastq.zst

Filename:           Results/02-readable-names/sample01-cond1-rep01.fastq.zst
Sequences:          1093957
Bases:              55791807
Mean-length:        51.00
Standard-deviation: 0.00
Min-length:         51
Max-length:         51
A:                  15928868 (28.55%)
C:                  12313956 (22.07%)
G:                  12227288 (21.92%)
T:                  15319751 (27.46%)
N:                  1944 (0.00%)
GC:                 24541244 (43.99%)
```

The **fastqc** command produces some basic quality graphs and plots showing summary statistics from a large and representative sample of our data. The output is not self-explanatory, so you will want to consult the FastQC documentation. You can either go to the FastQC website, or just run **fastqc** with no arguments (to open the FastQC GUI) and click on the `Help` menu.

When run with the name of a FASTQ file as a command-line argument, **fastqc** outputs a .zip file containing the results, and an HTML file that can be opened in any web browser to graphically view the results. Below is an example of running **fastqc** manually on one raw FASTQ file.

Note At the time of this writing, FastQC cannot read zstd compressed files directly. However it can read from the standard input (normally the terminal keyboard) by replacing the filename argument with "stdin:tag", where "tag" is any text we choose to describe the sample. Using a Unix pipe, we can direct the standard output (normally the terminal screen) of **zstdcat** to the standard input of **fastqc** as shown below. This piping technique is important to know, as it is useful with a wide variety of bioinformatics tools and compression tools.

```
# Create the output directory
shell-prompt: mkdir Results/03-qc-raw

# Run FastQC, putting the .zip and .html files in the output directory.
# The '\' at the end of the first line tells the shell that the command
# continues on the next line.
# "-o" tells FastQC to place output in the directory name that follows.
shell-prompt: zstdcat Results/02-readable-names/sample01-cond1-rep01.fastq.zst \
              | fastqc stdin:sample01-cond1-rep01 -o Results/03-qc-raw
null
Started analysis of stdin:sample01-cond1-rep01
Analysis complete for stdin:sample01-cond1-rep01

# Verify that the output files are where they should be
shell-prompt: ls Results/03-qc-raw
sample01-cond1-rep01_fastqc.html  sample01-cond1-rep01_fastqc.zip

# View the graphical FastQC report.
# Replace firefox with your favorite browser.
# Giving firefox a directory name as an argument should cause it
# to open a page with a list of the files in that directory.
# Click on any of the .html files to view the report for that sample.
shell-prompt: firefox Results/03-qc-raw/
```

This should show a page with a file list like the one in Figure 4.2.

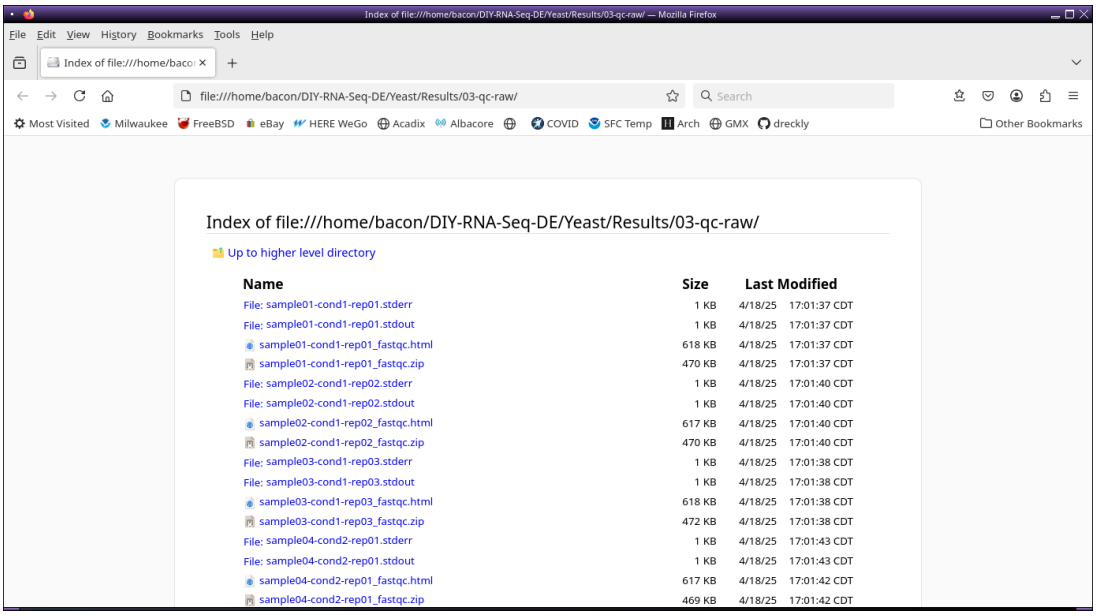

Figure 4.2: FastQC file listing

Then click on one of the .html files. This should bring you to a page like the one in Figure 4.3.

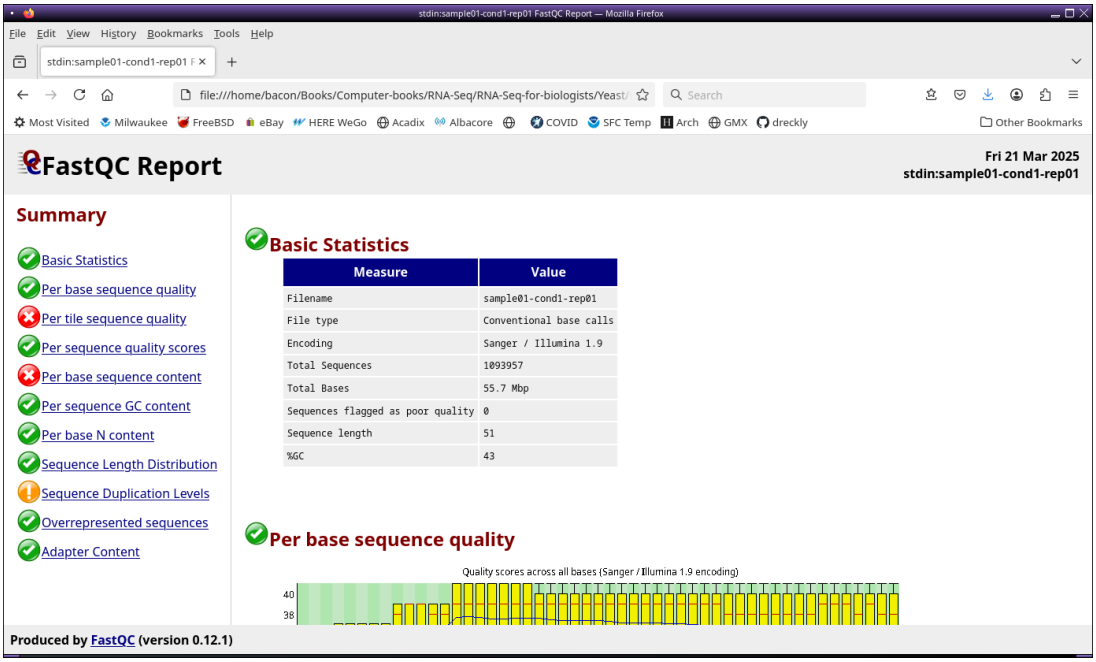

Figure 4.3: FastQC main page

The left panel shows a list of all the metrics and FastQC's assessment of the data quality for each of them. Green is high quality, orange is borderline, and red is low quality. Am orange or red alert for a given metric doesn't necessarily indicate a real problem. For example, significant sequence duplication is expected for RNA data, but FastQC will flag it anyway, because it doesn't know whether your sequences came from RNA or from a genome.

The DIY-RNA-Seq-DE repository provides a script for running **fastqc** on all the files created by **02-readable-names.sh**, using all available processors on your PC to complete the FastQC analyses as quickly as possible.

```
shell-prompt: ./03-qc-raw.sh

Darwin tarpon.acadix.biz 24.3.0 Darwin Kernel Version 24.3.0: Thu Jan  2 20:24:06 PST 2025; ↩
    root:xnu-11215.81.4~3/RELEASE_ARM64_T8103 arm64
FastQC v0.11.9
/Users/bacon/DIY-RNA-Seq-DE/Yeast
Running fastqc on each of the following:
Results/02-readable-names/sample04-cond2-rep03.fastq.zst
Results/02-readable-names/sample05-cond1-rep02.fastq.zst
Results/02-readable-names/sample03-cond2-rep02.fastq.zst
Results/02-readable-names/sample06-cond1-rep03.fastq.zst
Results/02-readable-names/sample02-cond2-rep01.fastq.zst
Results/02-readable-names/sample01-cond1-rep01.fastq.zst
Please wait...
Started analysis of stdin:sample01-cond1-rep01
Analysis complete for stdin:sample01-cond1-rep01
Started analysis of stdin:sample02-cond2-rep01
Analysis complete for stdin:sample02-cond2-rep01
Started analysis of stdin:sample03-cond2-rep02
Analysis complete for stdin:sample03-cond2-rep02
Started analysis of stdin:sample04-cond2-rep03
Analysis complete for stdin:sample04-cond2-rep03
Started analysis of stdin:sample05-cond1-rep02
Analysis complete for stdin:sample05-cond1-rep02
Started analysis of stdin:sample06-cond1-rep03
```

```
Analysis complete for stdin:sample06-cond1-rep03

# Run any web browser with the results directory as a command-line argument.
# This should show a file listing.  Click on any .html file to view results.
shell-prompt: firefox Results/03-qc-raw
```

```
# To save terminal output to a file:
shell-prompt: ./03-qc-raw.sh |& tee qc-raw.out

# If the syntax above doesn't work in your shell:
shell-prompt: ./03-qc-raw.sh 2>&1 | tee qc-raw.out
```

Note
Be sure to carefully review the output to the terminal for warnings, errors, and reasonable values. If you saved a copy of the terminal output by piping through **tee**, you can use tools such as **grep -i error file.out** to search for error messages, etc.

The sections below briefly describe each of the FastQC metrics and how to interpret them. For more details, see the FastQC documentation. This is mostly just informative, but the %GC content can be compared to known values for the organism as a basic check here.

4.7.1 Basic statistics

The basic statistics section provides a brief overview about the FASTQ file represented in this report, as shown in Table 4.1.

Measure	Value
Filename	sample01-cond1-rep01
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	1093957
Total Bases	55.7 Mbp
Sequences flagged as poor quality	0
Sequence length	51
%GC	43

Table 4.1: Basic statistics

4.7.2 Per base sequence quality

Figure 4.4 shows PHRED score statistics for each base in reads from one FASTQ file. Note that each column represents scores for that base in *all* reads, which likely number in the millions. The green zone (PHRED scores above 28, P(error) < 0.00158) is considered high quality by FastQC. The yellow zone (PHRED scores above 20, P(error) < 0.01) indicates acceptable quality for most purposes. The red zone indicates that we probably want to trim these bases off the reads.

The thin red lines indicate the median PHRED score at that base across all reads. The blue line tracks the means, rather than medians. Yellow boxes represent the interquartile range (containing scores between the 25th and 75th percentile). I.e., the middle 50% of the quality scores are within the yellow box. This means that if the yellow box dips into the red zone, then at least 25% of reads had low quality scores at that position. The whiskers are similar to the yellow boxes, but representing the 10th to 90th percentiles, so 80% of quality scores are within this range.

We can see in this graph that the sequencer was very confident about most of the bases read. However, a small percentage of bases at the 2nd, 3rd, and 4th positions have questionable quality scores. E.g., 10% of the reads had quality scores below 16 in positions 2 and 3. Hence, we might choose to trim the first four bases from all the reads in order to improve the overall read quality. On the other hand, the mean and median quality scores are good, so if a few incorrect bases are not a problem

for our downstream analysis (the analysis steps that follow), then we might choose to keep these bases and have longer reads to align. The **fastq-trim** command, discussed in Section 4.8, uses an augmented end-quality check which will detect the issue in positions 2 to 4, even though the first base is high quality. It will then trim off the 5' ends only in those reads where the quality is questionable.

Note We cannot just trim bases 2, 3, and 4, as this would change the DNA sequence of the read by making base 1 adjacent to base 5. We must always trim all the way to the beginning or end of a read.

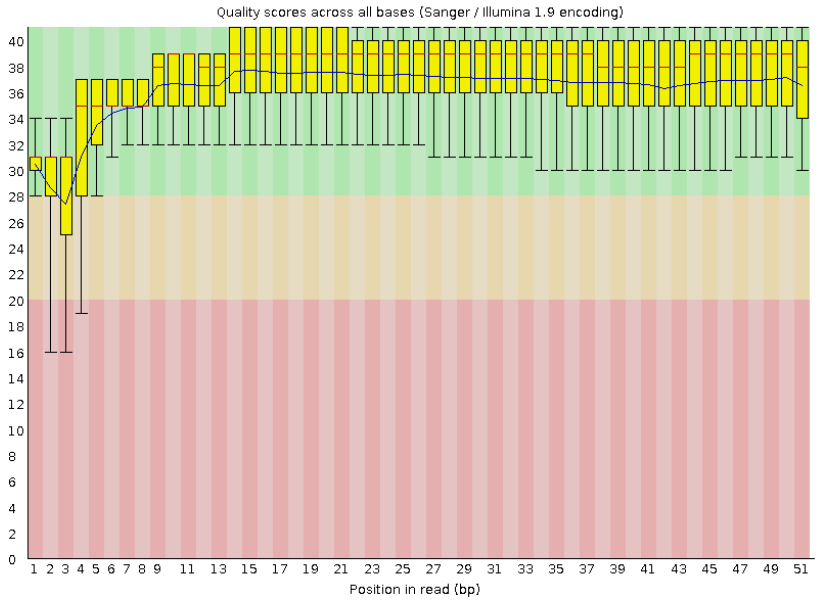

Figure 4.4: Raw read quality

4.7.3 Per tile sequence quality

Plots like the one shown in Figure 4.5 only appear if your reads came from an Illumina sequencer. It represents a map showing the quality of reads coming from each flow cell in the sequencer. Blue indicates the best quality, while red indicates the worst. Understanding this plot requires some knowledge of how Illumina sequencers work, which is beyond the scope of this text. For more information, consult the Illumina documentation, or talk to your sequencing specialist.

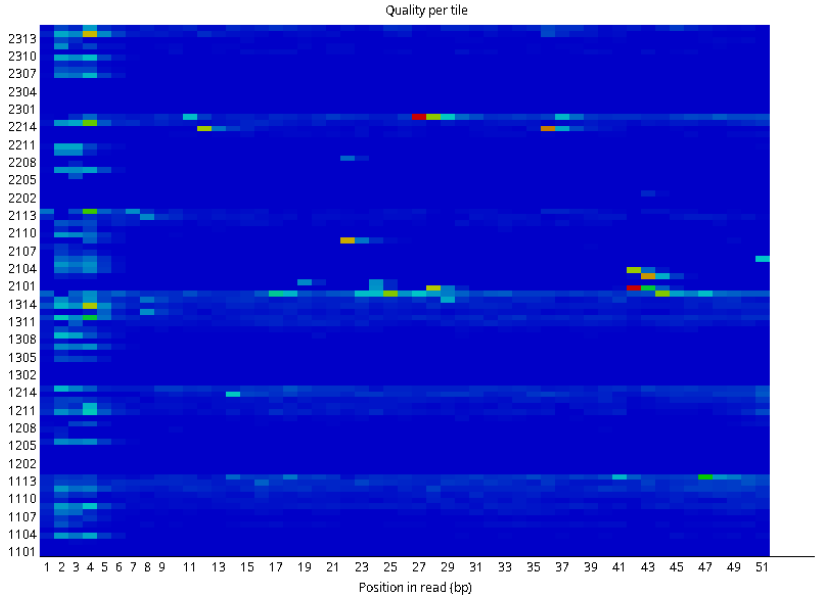

Figure 4.5: Per tile sequence quality

4.7.4 Per sequence quality scores

The graph in Figure 4.6 shows the statistical distribution of the average quality score for each read. Recall that quality scores are recorded for each base. This graph only represents the average score for all bases in each read, so some information is lost. In this graph, we see that over 200,000 reads had average quality scores of 38, and another 200,000 had average scores of 39. About 30,000 reads had average quality scores of 30.

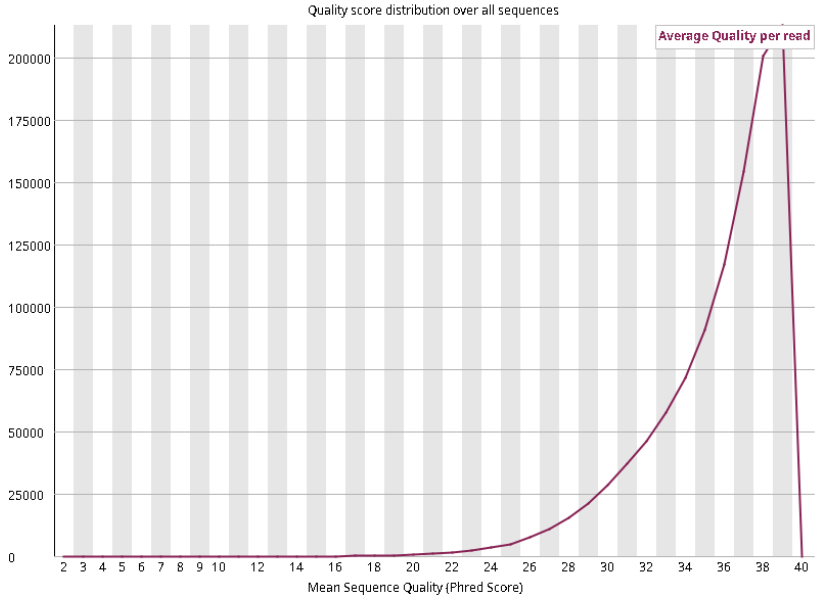

Figure 4.6: Per sequence quality scores

4.7.5 Per base sequence content

Base content (GC vs AT) is a known quantity for many species and the content found in the reads should match it closely. This provides a quick litmus test for data quality. Bias in the first 10 to 20 bases, such as that shown in Figure 4.7, is common. It is not cause for alarm, as long as the PHRED scores indicate good reads. This bias is due to the fact that the reads come from DNA or RNA that was fragmented by restriction enzymes, which bind to specific motifs (patterns) in the sequence. Hence, the 5' end of each fragment in our library is less random than the rest of the bases. We can see from this graph that the restriction enzyme used strongly favors a thymine at position 7.

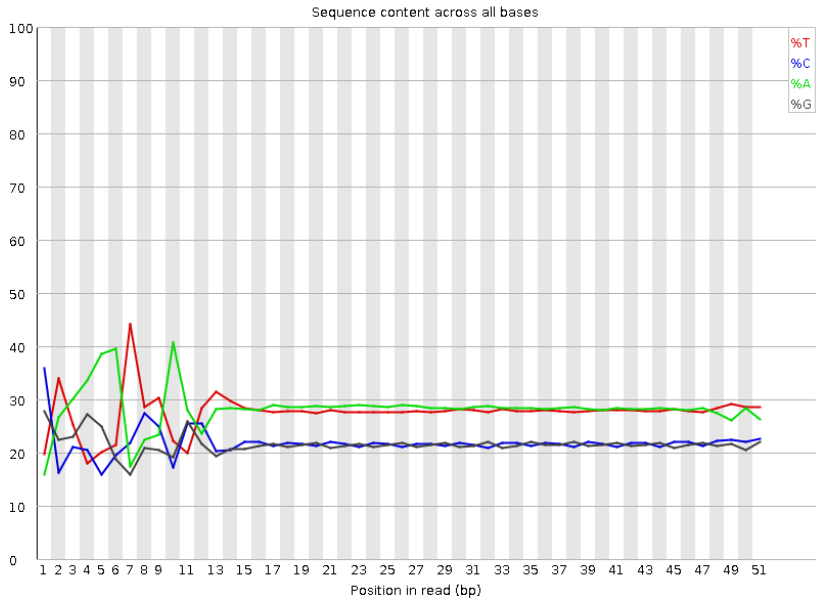

Figure 4.7: Base content

4.7.6 Per sequence GC content

Per sequence GC content shows the *distribution* of GC content among all the reads. I.e., it shows how many reads contain each possible GC content. We see here that most of the reads have a GC content around 44%. Some have GC content as low as 20% or as high as 68%. This should not be too surprising, given that these are short reads (only 51 bases), and repetitive sequences are common in most genomes. The mean GC content should match what is known for the species under study.

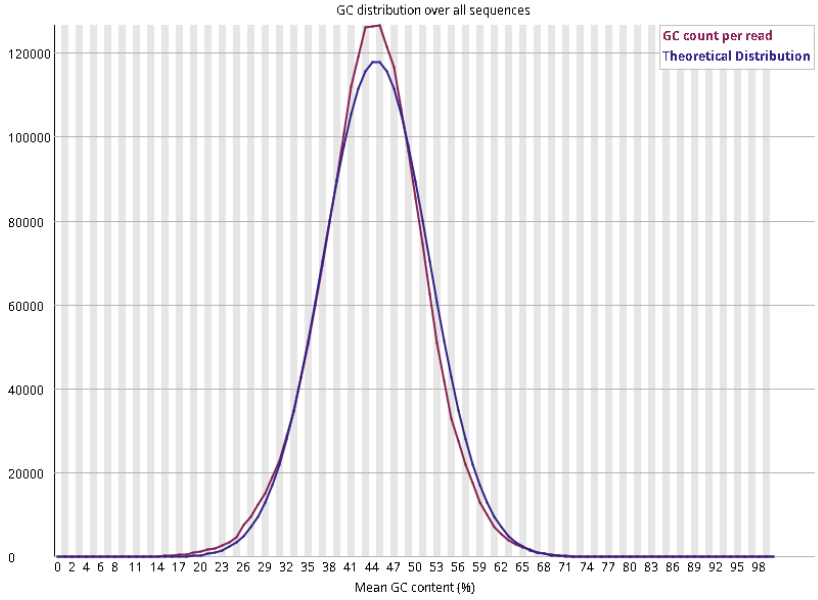

Figure 4.8: Per sequence GC content

4.7.7 Per base N content

N in a sequence, as opposed to A, C, G, or T/U represents an unknown base. We see in this graph that the N content is very low across the board, which is what we expect from a successful sequencing run.

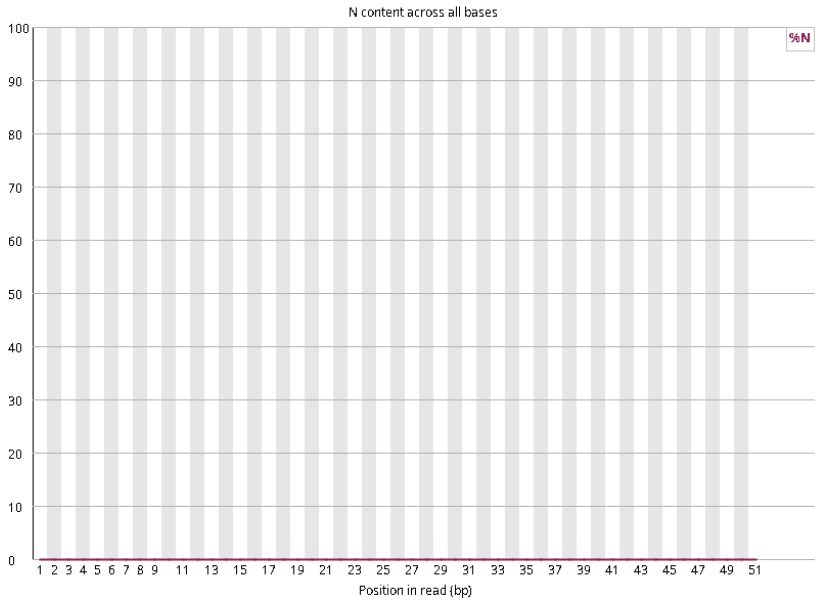

Figure 4.9: Per base N content

4.7.8 Sequence length distribution

Sequence length distribution shows how many sequences there were of each length among all the reads. We see here that all of the more than 1,000,000 reads were exactly 51 bases, which is what we expect from raw FASTQ files. After trimming, covered

in Section 4.8, some of the reads will be shortened and this graph will look more like a curve than a pyramid.

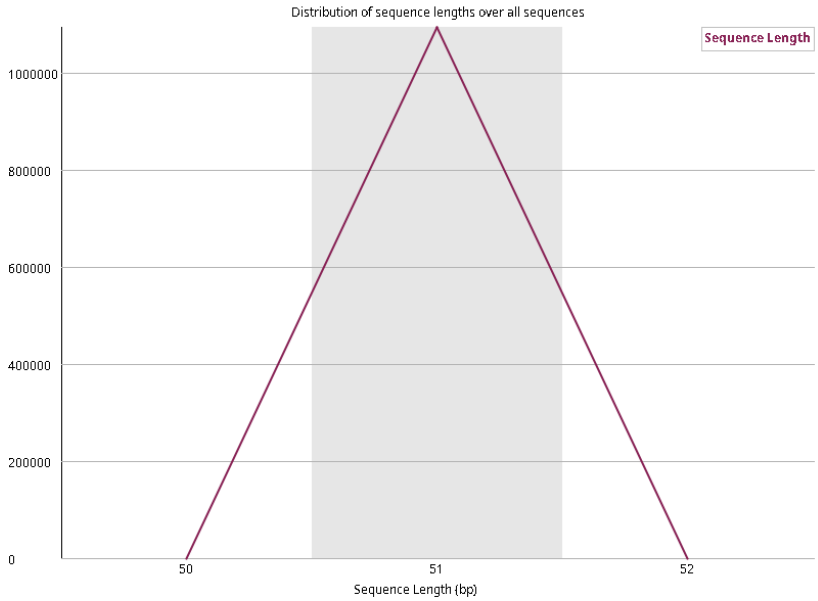

Figure 4.10: Sequence length distribution

4.7.9 Sequence duplication levels

Sequence duplication levels show the percentage of sequences duplicated (replicated, really, since "duplicated" means exactly two copies) N times. We see here that about 55% of all reads have one suspected replicate, about 12% have two, and so on.

This graph also indicates at the top that 100 - 67.52 = 32.48% of all reads are suspected replicates.

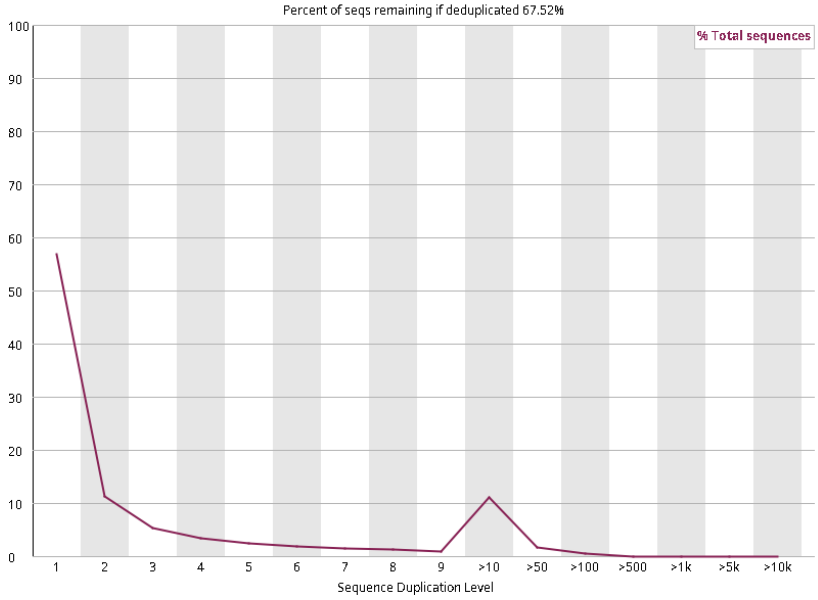

Figure 4.11: Sequence duplication levels

4.7.10 Overrepresented sequences

This metric flags sequences that appear in the data more often than they would by chance, indicating likely duplicates. This file contained no overrepresented sequences according to FastQC. Table 4.2 shows an example from another data set.

Sequence	Count	Percentage	Possible Source
CCGCTGTATTACTCAGGCTGCACT	500476	0.7987309575323511	No Hit
CGCTGTATTACTCAGGCTGCACTG	321778	0.5135392107770299	No Hit
GTCAGGTGTTCTGAGGCTTAGAGA	201896	0.3222144226735179	No Hit
GGTTATTTCATTTCTCTTTCAGCG	192603	0.3073833282986665	No Hit

Table 4.2: Overrepresented sequences

4.7.11 Adapter content

The adapter content graph shows the percentage of reads containing adapters starting at each position in the reads. The 5' adapters are almost always removed before the sequencing center gives us our FASTQ files, but there may be some 3' adapters present in fragments that were shorter than the read length.

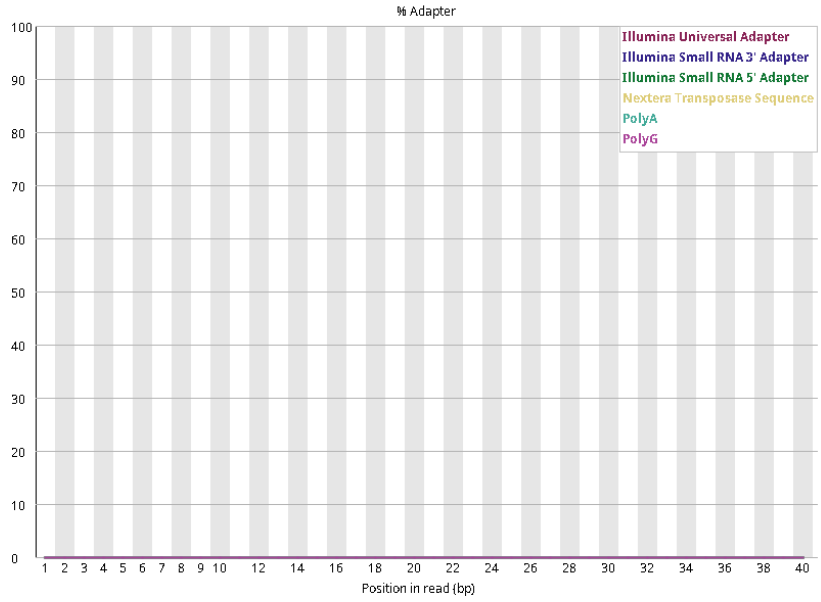

Figure 4.12: Adapter content

According to this plot, these raw reads are extremely clean. Seeing no measurable adapter content makes me wonder if something is wrong, so I'm inclined to corroborate this result. The **fastq-scum** command, part of the **fastq-trim** package, will show all lines in a FASTQ file containing common adapters (see Figure 4.13).

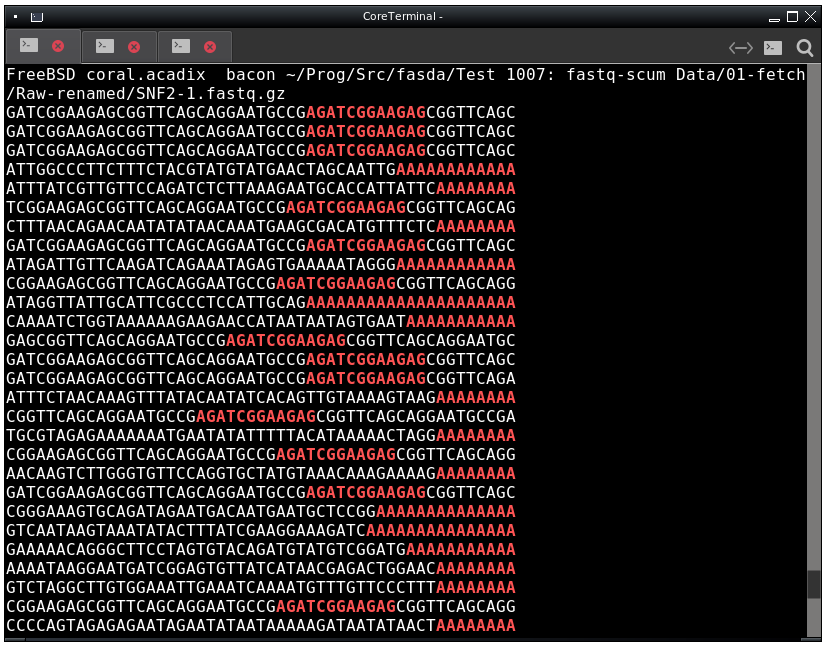

Figure 4.13: Inspecting data with fastq-scum

Piping the output through **wc -l** will give us the number of reads containing a known adapter or poly-A tail. We can then compare with the total number of lines in the file, determined using **zstdcat** and **wc -l**. Doing this, we see that there are indeed some adapter sequences in the file, but only 1 in 3,492 reads (0.02%) contains one. Hence, it is no surprise that the FastQC graph does not show them. While the adapter count is not zero, it's too low to show up on a low-resolution graph like this one.

```
shell-prompt: fastq-scum Results/02-readable-names/sample01-cond1-rep01.fastq.zst | wc -l
    1253
shell-prompt: zstdcat Results/02-readable-names/sample01-cond1-rep01.fastq.zst | wc -l
 4375828

# Use the standard Unix "bc" calculator to compute frequency of adapters
shell-prompt: bc
4375828/1253
3492.28092577813248204309
```

4.7.12 MultiQC

As mentioned in Section 2.3, MultiQC is considered optional here, because it depends on over 200 other software packages and therefore takes a long time to install from source via dreckly. However, a very simple script is included in the DIY-RNA-Seq-DE repository for running MultiQC, if you so desire.

Note MultiQC requires the locale to use a UTF-8 character set, as it uses special characters in the output. Hence, we must set the LANG and LC_ALL environment variables if they are not set to a UTF-8 character set by default.

```
shell-prompt: mkdir -p Results/04-multiqc-raw
shell-prompt: env LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8 \
    multiqc --outdir Results/04-multiqc-raw Results/03-qc-raw
```

The DIY-RNA-Seq-DE provides a script for running MultiQC automatically:

```
shell-prompt: ./04-multiqc-raw.sh
multiqc, version 1.25.2

/// MultiQC  v1.25.2

     version_check | MultiQC Version v1.28 now available!
       file_search | Search path: /home/bacon/Books/Computer-books/RNA-Seq/DIY-RNA-Seq-DE/ ↩
           Yeast/Results/03-qc-raw
         searching | 100% 24/24
             fastqc | Found 6 reports
     write_results | Existing reports found, adding suffix to filenames. Use '--force' to  ↩
         overwrite.
     write_results | Data        : Results/04-multiqc-raw/multiqc_data_1
     write_results | Report      : Results/04-multiqc-raw/multiqc_report_1.html
           multiqc | MultiQC complete

shell-prompt: firefox Results/04-multiqc-raw/multiqc_report_1.html
```

```
# To save terminal output to a file:
shell-prompt: ./04-multiqc-raw-.sh |& tee multiqc-raw.out

# If the syntax above doesn't work in your shell:
shell-prompt: ./04-multiqc-raw.sh 2>&1 | tee multiqc-raw.out
```

Note
Be sure to carefully review the output to the terminal for warnings, errors, and reasonable values. If you saved a copy of the terminal output by piping through **tee**, you can use tools such as **grep -i error file.out** to search for error messages, etc.

A couple of example MultiQC plots are shown below. Figure 4.14 shows how MultiQC represents sequence counts for all samples as a bar chart. Figure 4.15 shows the sequence quality distributions for all samples combined into a single graph.

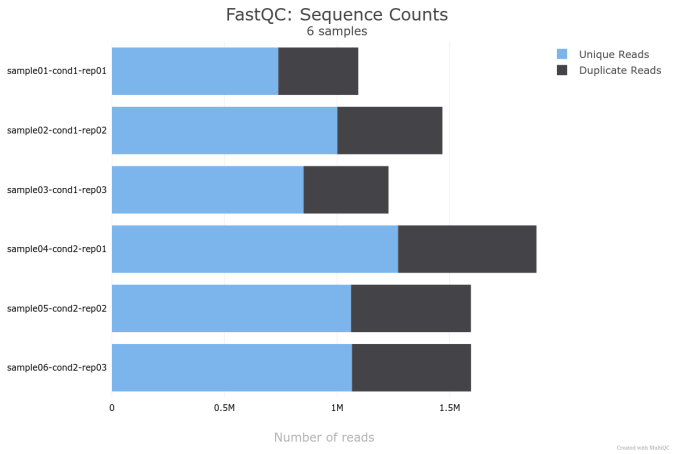

Figure 4.14: MultiQC sequence counts

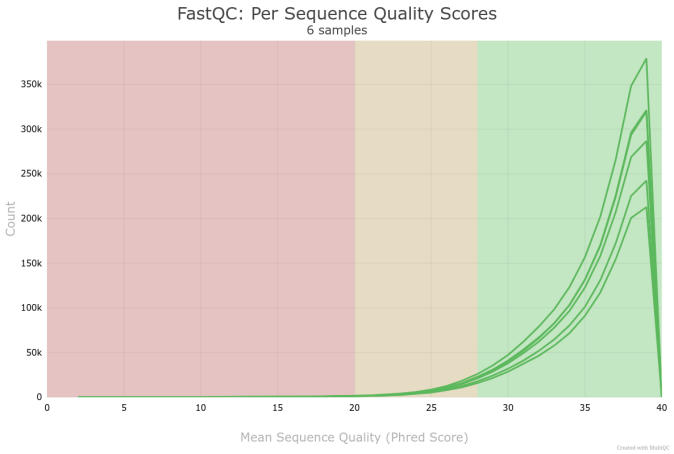

Figure 4.15: MultiQC per sequence quality

4.7.13 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. How can we easily interpret the quality scores we see in the raw data files?
2. Show a command that prints summary statistics about the file `sample01-cond1-rep3.fq.xz`. How do we install this command?
3. Show a command that produces summary quality information about the file `sample01-cond1-rep3.fq.xz`, placing results in `Results/Raw-QC`. Hint: You may need to review Section 4.5.
4. Show a command for viewing FastQC results in the directory `Results/Raw-QC`.
5. What does a red flag in the FastQC left panel indicate?
6. What does the thin red line indicate in the per-base quality graph?
7. What do the whiskers indicate in the per-base quality graph?
8. What does the per-tile quality graph tell us?
9. What does the per sequence quality scores graph tell us?
10. What does the per base sequence content graph tell us? What causes the bias often seen at the 5' end of each read?
11. What does the per sequence GC content graph tell us?
12. What does the per base N content graph tell us?
13. What does the sequence length distribution graph tell us? What do we expect to see here for typical raw RNA-Seq data?
14. What does the sequence length distribution level graph tell us? What do we expect to see here for RNA-Seq data?
15. What does the overrepresented sequences graph tell us?
16. What does the adapter content graph tell us? What do we expect to see here and why?
17. What does MultiQC do?

4.8 Adapter trimming

As noted earlier, the sequencing process involves appending artificial sequences, called *adapters*, to both ends of a DNA/RNA fragment. These adapters should be removed so that they do not affect subsequent alignment or other analysis steps.

There are several existing tools for adapter removal, such as **cutadapt** and **Trimmomatic**. While we found **cutadapt** to be an excellent tool, I decided to develop a new tool, called **fastq-trim**, as an experiment to see how much performance gain was possible using well-optimized C code. The results were quite positive, and are available, along with the code, at https://github.com/-auerlab/fastq-trim. In summary, **fastq-trim** runs more than twice as fast as **cutadapt**, four times as fast as **trimmomatic**, and uses a small fraction of the memory of either, while producing results nearly identical to those of cutadapt. Figure 4.16 shows the performance metrics. "Wall time" represents actual run time according to the clock on the wall. "CPU util" is average CPU utilization (100% = 1 CPU fully utilized). The fastq-trim CPU utilization includes CPU time used by compression tools to decompress the input files and compress the output files. Resident memory is the peak amount of electronic memory (RAM) used (fastq-trim only, not compression tools) in MiB. Virtual memory is RAM + swap space (temporary disk storage for code and data that is not being actively accessed).

```
                CPU     Wall    Virtual Resident
                util    time    memory  memory
Fastq-trim      260%    5.42    13      2
Cutadapt 1-core 140%    22.37   46      30
Cutadapt 2-core 310%    13.30   174     120
Trimmomatic     150%    21.03   3473    740
```

Figure 4.16: Fastq-trim performance

The default alignment algorithm used by **fastq-trim** is very similar to the one used by cutadapt. Comparison of trimming results for a sample with one million reads, we saw only 84 differences between **fastq-trim** and **cutadapt** output (0.0085% disagreement). Disagreement between **cutadapt** and **Trimmomatic**, two of the most popular trimming tools, was much greater.

In addition to adapter removal, it is common to remove bases with low PHRED scores at both the 5' and 3' ends of the read, along with poly-A tails. While some trimmers handle this with additional trimming steps, **fastq-trim** performs adapter removal, poly-A removal, and end-clipping together in a single pass.

Some trimming tools attempt to automatically detect which adapter sequences were used. While some people might view this as an attractive convenience, we adhere to the "know your data" principal, rather than use "black box" tools, where we don't know what is happening to our data. Our approach is to use a simple tool included with **fastq-trim** called **fastq-scum**, which visually highlights known adapter sequences, as well as poly-A tails, as shown in Figure 4.17. We then explicitly specify the adapter sequence(s) revealed by **fastq-scum** in the **fastq-trim** commands. By visually inspecting the data this way before trimming, we gain a better understanding of the data that were collected and what to expect during processing.

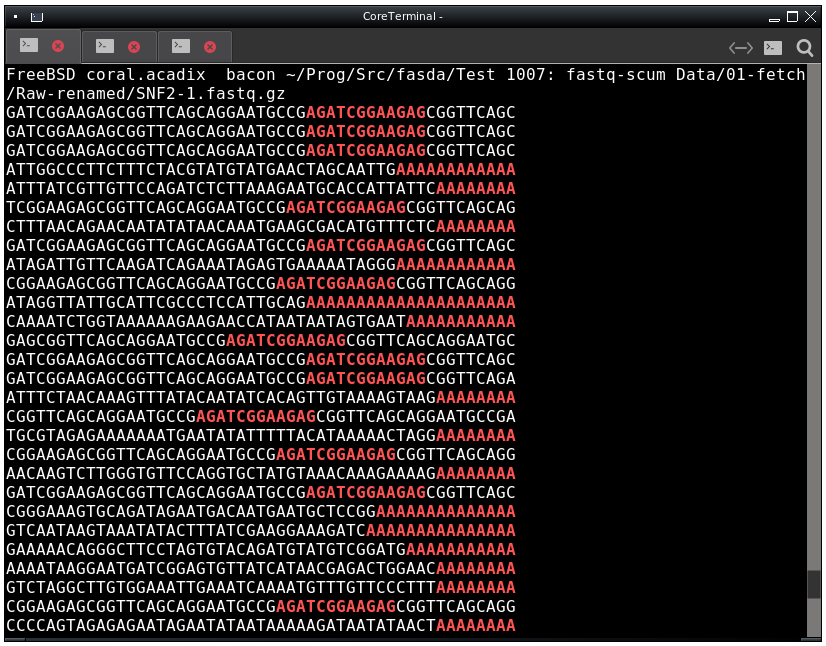

Figure 4.17: Inspecting data with fastq-scum

Using the information obtained from **fastqc** and **fastq-scum**, we can construct an individualized trim command for removing adapters, poly-A tails, and low quality bases.

The command below removes the adapter shown by **fastq-scum** above, along with any sequence downstream of it (toward the 3' end) of the read. If the read ends with a 5' portion of the adapter of three bases or more, that sequence is removed as well. If more than 10% of the bases differ from the adapter, the sequence is assumed to be natural and not removed. Immediately after adapter removal, it removes a poly-A tail of length three or more from the 3' end. Following poly-A removal, 5' and 3' end sequences of bases with an average quality score below 20 are trimmed. If the resulting trimmed read has fewer than 30 bases remaining, then the entire read is discarded (not included in the trimmed output).

```
# Create the output directory
shell-prompt: mkdir Results/04-trim

# Manually run fastq-trim on sample 1
# Run fastq-trim under the Unix "time" command to measure how long it takes
shell-prompt: time fastq-trim --3p-adapter1 AGATCGGAAGAG --polya-min-length 3 \
    --min-match 3 --max-mismatch-percent 10 --min-qual 20 --min-length 30 \
    Results/02-readable-names/sample01-cond1-rep01.fastq.zst \
    Results/04-trim/sample01-cond1-rep01-trimmed.fastq.zst

*** FASTQ TRIM ***

  Minimum match:     3
  Minimum quality:   20
  Minimum length:    30
  Phred base:        33
  Adapter matching:  Smart
  Maximum mismatch:  10%
  Filename:          Results/02-readable-names/sample01-cond1-rep01.fastq.zst
  Read-type:         Single
  Adapter:           AGATCGGAAGAG

Read: 1000000  Adapter: 29743  Poly-A: 7133  Q < 20: 59529  Len < 30: 2476

Reads:                              1093957
Reads with adapters:                  32462 (2%)
```

```
Reads with Poly-As:                      7756 (0%)
Bases with Q < 20:                     303890 (0%)
Reads with low Q bases removed:         64963 (5%)
Reads < 30 bases after trimming:         2684 (0%)
Mean adapter position:                     46
Mean read length:                          51

# Output from time command
# fastq-trim used 7 seconds of CPU time (including zstd [un]compression),
# .429 seconds of operating system services (mostly file I/O),
# and 3.88 seconds of wall time.  CPU time exceeds wall time because
# more than one CPU was used.
7.386u 0.429s 0:03.88 201.0%    525+168k 0+63io 0pf+0w
```

Note

If you want to use a different compression tool with **fastq-trim**, simply change the filename extension on the input or output file. E.g., if your input files are compressed with **xz**, then they should have a ".xz" extension, and we would use `Results/02-readable-names/sample01-cond1-rep01.fastq.xz`. If you want to compress the output with **gzip** instead of **zstd**, use `Results/04-trim/sample01-cond1-rep01-trimmed.fastq.gz`.

All of the tools we developed, available at https://github.com/auerlab/ and https://github.com/outpaddling/ share this feature: They support all popular compression tools and automatically use the appropriate tool based on the filename extension (".gz", ".bz2", ".lz4", ".xz", or ".zst"). Choose your compression tools based on the importance of saving disk space and on which tools will work for the next stage of the pipeline (FastQC and read mapping in this case).

As with other pipeline stages, the DIY-RNA-Seq-DE repository contains a script for trimming all the raw files, using as many CPUs as possible on your computer. Below is a portion of the output from this script, with some data in the center removed for brevity.

```
shell-prompt: ./05-trim.sh
FreeBSD moray.acadix.biz 14.2-RELEASE-p1 FreeBSD 14.2-RELEASE-p1 GENERIC amd64
fastq-trim 0.1.3.11
/home/bacon/Books/Computer-books/RNA-Seq/DIY-RNA-Seq-DE/Yeast
Running fast-trim on each of the following:
Results/02-readable-names/sample01-cond1-rep01.fastq.zst
Results/02-readable-names/sample02-cond1-rep02.fastq.zst
Results/02-readable-names/sample03-cond1-rep03.fastq.zst
Results/02-readable-names/sample04-cond2-rep01.fastq.zst
Results/02-readable-names/sample05-cond2-rep02.fastq.zst
Results/02-readable-names/sample06-cond2-rep03.fastq.zst

*** FASTQ TRIM ***

  Minimum match:     3
  Minimum quality:   20
  Minimum length:    30
  Phred base:        33
  Adapter matching:  Smart
  Maximum mismatch:  10%
  Filename:          Results/02-readable-names/sample01-cond1-rep01.fastq.zst
  Read-type:         Single
  Adapter:           AGATCGGAAGAG

Reads:                                1093957
Reads with adapters:                    32462 (2%)
Reads with Poly-As:                      7756 (0%)
Bases with Q < 20:                     303890 (0%)
Reads with low Q bases removed:         64963 (5%)
Reads < 30 bases after trimming:         2684 (0%)
```

```
Mean adapter position:                      46
Mean read length:                           51

[ Results for other samples omitted for brevity ]

*** FASTQ TRIM ***

  Minimum match:     3
  Minimum quality:   20
  Minimum length:    30
  Phred base:        33
  Adapter matching:  Smart
  Maximum mismatch:  10%
  Filename:          Results/02-readable-names/sample06-cond2-rep03.fastq.zst
  Read-type:         Single
  Adapter:           AGATCGGAAGAG



Reads:                                1594695
Reads with adapters:                    46688 (2%)
Reads with Poly-As:                     13917 (0%)
Bases with Q < 20:                     435989 (0%)
Reads with low Q bases removed:         92568 (5%)
Reads < 30 bases after trimming:         3439 (0%)
Mean adapter position:                     47
Mean read length:                          51
```

```
# To save terminal output to a file (C shell):
shell-prompt: ./05-trim.sh |& tee trim.out

# If the syntax above doesn't work in your shell (Bourne shell):
shell-prompt: ./05-trim.sh 2>&1 | tee trim.out
```

Note

Be sure to carefully review the output to the terminal for warnings, errors, and reasonable values. If you saved a copy of the terminal output by piping through **tee**, you can use tools such as **grep -i error file.out** to search for error messages, etc.

The yeast data we are using for this example are single-ended. When processing paired-end data, the situation is slightly more complex. The filenames for paired-end reads traditionally contain "R1" for forward (5') reads and "R2" for reverse (3') reads. To use **fastq-trim** on paired-end data, we must specify two input files, two output files, and two adapters, since the adapters in theory could actually be different for the two files (though I have not seen an example of this). The **fastq-trim** command will keep the two output files properly mated. Any time an entire read is discarded from one file (due to quality issues), the corresponding read (the mate) in the other file is removed as well, regardless of its quality metrics. This is necessary to keep the files in sync, since the position of a read within the file is the only way to pair it with its mate in the other file.

```
shell-prompt: fastq-trim --min-qual 24 --polya-min-length 4 \
    --3p-adapter1 AGATCGGAAGAG --3p-adapter2 AGATCGGAAGAG \
    sample1-rep1-time1-R1.fastq.zst sample1-rep1-time1-R1-trimmed.fastq.zst \
    sample1-rep1-time1-R2.fastq.zst sample1-rep1-time1-R2-trimmed.fastq.zst
```

The DIY-RNA-Seq-DE repository provides a script for trimming all of the raw read files.

```
shell-prompt: ./05-trim.sh

Darwin tarpon.acadix.biz 24.3.0 Darwin Kernel Version 24.3.0: Thu Jan  2 20:24:06 PST 2025; ↩
     root:xnu-11215.81.4~3/RELEASE_ARM64_T8103 arm64
fastq-trim 0.1.3
```

```
/Users/bacon/DIY-RNA-Seq-DE/Yeast
Running fast-trim on each of the following:
Results/02-readable-names/sample04-cond2-rep03.fastq.zst
Results/02-readable-names/sample05-cond1-rep02.fastq.zst
Results/02-readable-names/sample03-cond2-rep02.fastq.zst
Results/02-readable-names/sample06-cond1-rep03.fastq.zst
Results/02-readable-names/sample02-cond2-rep01.fastq.zst
Results/02-readable-names/sample01-cond1-rep01.fastq.zst
Please wait...

*** FASTQ TRIM ***

  Minimum match:     3
  Minimum quality:   20
  Minimum length:    30
  Phred base:        33
  Adapter matching:  Smart
  Maximum mismatch:  10%
  Filename:          Results/02-readable-names/sample01-cond1-rep01.fastq.zst
  Read-type:         Single
  Adapter:           AGATCGGAAGAG

Reads:                              1093957
Reads with adapters:                  32462 (2%)
Reads with Poly-As:                    7756 (0%)
Bases with Q < 20:                   303890 (0%)
Reads with low Q bases removed:       64963 (5%)
Reads < 30 bases after trimming:       2684 (0%)
Mean adapter position:                   46
Mean read length:                        51

[ Output omitted for brevity ]

*** FASTQ TRIM ***

  Minimum match:     3
  Minimum quality:   20
  Minimum length:    30
  Phred base:        33
  Adapter matching:  Smart
  Maximum mismatch:  10%
  Filename:          Results/02-readable-names/sample06-cond1-rep03.fastq.zst
  Read-type:         Single
  Adapter:           AGATCGGAAGAG

Reads:                              1228207
Reads with adapters:                  36009 (2%)
Reads with Poly-As:                   11010 (0%)
Bases with Q < 20:                   347302 (0%)
Reads with low Q bases removed:       73271 (5%)
Reads < 30 bases after trimming:       2783 (0%)
Mean adapter position:                   47
Mean read length:                        51
```

4.8.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. What is the goal of read trimming in an RNA-Seq analysis?
2. What is the disadvantage of using a trimming tool that automatically detects which adapter sequences were used? What is the advantage?
3. What compression tools can be used implicitly with **fastq-trim** input and output files?
4. Show a **fastq-trim** command that will trim reads in the paired-end files `sample01-cond2-rep05-R1.fastq.xz` and `sample01-cond2-rep05-R2.fastq.xz`, saving the output to `sample01-cond2-rep05-R1-trimmed.fastq.gz` and `sample01-cond2-rep05-R2-trimmed.fastq.gz`. The 3' adapter begins with AGATCGGAAGAG. Trim sequences that match at least the first four bases of the adapter with no more than 15% differences. Trim ends with PHRED scores below 25 and poly-A tails of length five or more. Discard reads less than 20 bases in length after trimming. Save a copy of all terminal output to `trim.out`, assuming you are using a Bourne shell derivative.

4.9 Post-trim quality check

After trimming, we repeat the FastQC quality checks on the trimmed data, to verify that any quality issues we noticed have been eliminated. The example below shows how to run the post-trim quality checks using the script provided in the DIY-RNA-Seq-DE/Yeast repository.

```
shell-prompt: ./06-qc-trimmed.sh

Darwin tarpon.acadix.biz 24.3.0 Darwin Kernel Version 24.3.0: Thu Jan  2 20:24:06 PST 2025; ↩
     root:xnu-11215.81.4~3/RELEASE_ARM64_T8103 arm64
FastQC v0.11.9
/Users/bacon/DIY-RNA-Seq-DE/Yeast
Running fastqc on each of the following:
Results/05-trim/sample06-cond1-rep03-trimmed.fastq.zst
Results/05-trim/sample01-cond1-rep01-trimmed.fastq.zst
Results/05-trim/sample05-cond1-rep02-trimmed.fastq.zst
Results/05-trim/sample04-cond2-rep03-trimmed.fastq.zst
Results/05-trim/sample03-cond2-rep02-trimmed.fastq.zst
Results/05-trim/sample02-cond2-rep01-trimmed.fastq.zst
Please wait...
Started analysis of stdin:sample01-cond1-rep01-trimmed
Analysis complete for stdin:sample01-cond1-rep01-trimmed
Started analysis of stdin:sample02-cond2-rep01-trimmed
Analysis complete for stdin:sample02-cond2-rep01-trimmed
Started analysis of stdin:sample03-cond2-rep02-trimmed
Analysis complete for stdin:sample03-cond2-rep02-trimmed
Started analysis of stdin:sample04-cond2-rep03-trimmed
Analysis complete for stdin:sample04-cond2-rep03-trimmed
Started analysis of stdin:sample05-cond1-rep02-trimmed
Analysis complete for stdin:sample05-cond1-rep02-trimmed
Started analysis of stdin:sample06-cond1-rep03-trimmed
Analysis complete for stdin:sample06-cond1-rep03-trimmed

shell-prompt: firefox Results/06-qc-trimmed
```

```
# To save terminal output to a file:
shell-prompt: ./06-qc-trimmed-.sh |& tee qc-trimmed.out

# If the syntax above doesn't work in your shell:
shell-prompt: ./06-qc-trimmed.sh 2>&1 | tee qc-trimmed.out
```

Note
Be sure to carefully review the output to the terminal for warnings, errors, and reasonable values. If you saved a copy of the terminal output by piping through **tee**, you can use tools such as **grep -i error file.out** to search for error messages, etc.

Most of the FastQC metrics look basically the same as those for the raw data, since the raw reads were quite clean to begin with. A few notable differences are shown below.

4.9.1 Basic statistics

FastQC output on the trimmed sample 1 FASTQ file shows a slight drop in total sequences (1093957 to 1091273) and total bases (55.7 Mbp to 55.2 Mbp), and sequence length now shows "30-51" instead of "51". So, 2,684 reads were removed entirely due to quality issues, and some of the remaining reads had ends trimmed off due to low quality scores, adapters, or poly-A tails.

Measure	Value
Filename	sample01-cond1-rep01
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	1091273
Total Bases	55.2 Mbp
Sequences flagged as poor quality	0
Sequence length	30-51
%GC	43

Table 4.3: Basic statistics

4.9.2 Per base sequence quality quality

The **fastq-trim** command trimmed some low-quality bases from the 5' end of some reads to bring up the average scores, as shown in Figure 4.18. Compare this to Figure 4.4, which shows some low-quality bases near the 5' end.

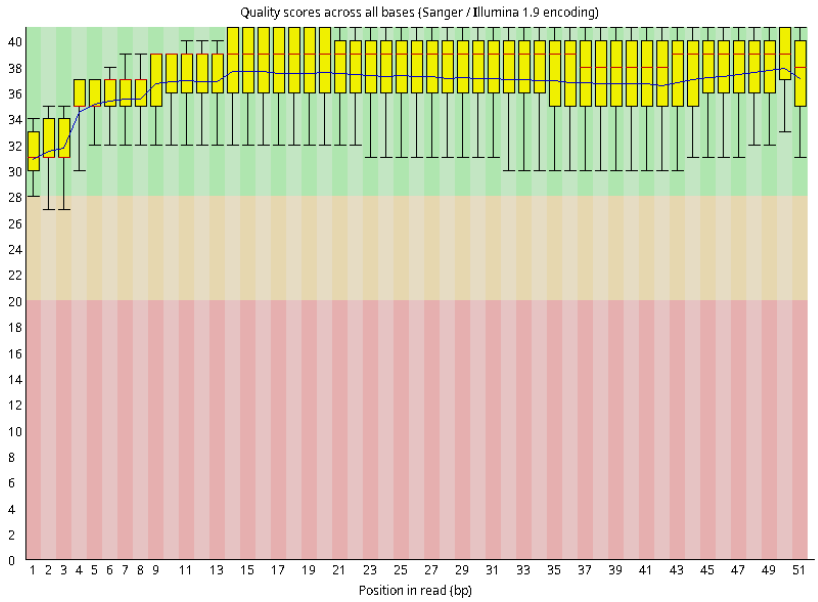

Figure 4.18: Trimmed read quality

4.9.3 Sequence length distribution

Figure 4.19 also shows a range of read lengths from 30 to 51. Recall that the raw sequence length distribution was a pyramid with a peak at 51 bp (all reads were exactly 51 bases).

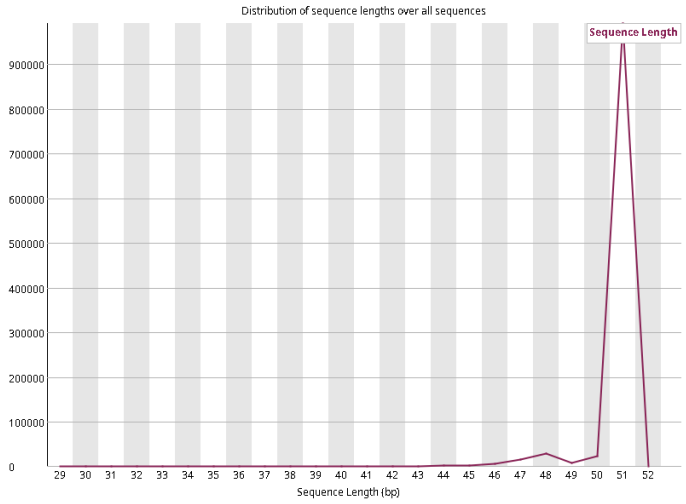

Figure 4.19: Sequence length distribution

4.9.4 MultiQC

The DIY-RNA-Seq-DE provides a script for running MultiQC on the trimmed reads, which again, is optional:

```
shell-prompt: ./07-multiqc-trimmed.sh
multiqc, version 1.25.2

/// MultiQC  v1.25.2

     version_check | MultiQC Version v1.28 now available!
       file_search | Search path: /home/bacon/Books/Computer-books/RNA-Seq/DIY-RNA-Seq-DE/ ↩
           Yeast/Results/06-qc-trimmed
         searching | 100% 24/24
             fastqc | Found 6 reports
     write_results | Existing reports found, adding suffix to filenames. Use '--force' to  ↩
         overwrite.
     write_results | Data        : Results/04-multiqc-raw/multiqc_data_1
     write_results | Report      : Results/04-multiqc-raw/multiqc_report_1.html
           multiqc | MultiQC complete

shell-prompt: firefox Results/07-multiqc-trimmed/multiqc_report_1.html
```

```
# To save terminal output to a file:
shell-prompt: ./07-multiqc-trimmed.sh |& tee multiqc-trimmed.out

# If the syntax above doesn't work in your shell:
shell-prompt: ./07-multiqc-trimmed.sh 2>&1 | tee multiqc-trimmed.out
```

Note
Be sure to carefully review the output to the terminal for warnings, errors, and reasonable values. If you saved a copy of the terminal output by piping through **tee**, you can use tools such as **grep -i error file.out** to search for error messages, etc.

4.9.5 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

---

1. What kind of changes should we expect to see in the FastQC basic statistics post-trim results, as compared with the pre-trim results, assuming fixed-length reads?
2. What kind of changes should we expect to see in the FastQC read length distribution post-trim results, as compared with the pre-trim results, assuming fixed-length reads?
3. What kind of changes should we expect to see in the FastQC per base read quality post-trim results, as compared with the pre-trim results?

4.10 Reference genome and transcriptome

4.10.1 What's a reference?

In order to map our reads (determine the location in the genome from which they were transcribed), we need a *reference* genome or transcriptome. A reference genome represents the entire DNA sequence of every chromosome in a hypothetical individual. A reference transcriptome represents all of the known coding sequences of a hypothetical individual, and is a small fraction of the size of the genome for most species. Since there are differences in DNA between any two individuals, and even differences in DNA between cells within an individual (due to mutations that occur in some cells and not others), many of our reads will not match the reference exactly. The reference sequence is designed in a way to maximize the likelihood of matching any individual, but using the most common bases (the consensus) among all individuals sampled.

One of the most common types of differences is the *single nucleotide polymorphism* (SNP), where sequences are the same length, but individual bases differ:

```
ATAGCTAGCTAGCTGATCGTAGCTAGCTAGTCAGCTGATCATCGGTCGATCTAGCTAGCTAGCTAG
ATAGCTAGCTAGCTGATCCTAGCTAGCTAGTCAGCTGATCATCGATCGATCTAGCTAGCTAGCTAG
                  ^                           ^
            Single nucleotide polymorphisms (SNPs)
```

Other differences include *insertions*, where one or more extra bases are inserted in the middle of a sequence, and *deletions*, where one or more bases are removed within a sequence. Insertions and deletions are really the same thing viewed from different angles, because they are always relative from one sample to another. An insertion in a sequence from sample A relative to sample B is a deletion in sequence B relative to sample A. The two terms are often combined and called *indels*.

```
# This sequence has an insertion of GCT relative to the one below
TAGCTAGCTAGCTAGCTAGCTGATCAGCTGATCGTAGCTAGTCGATCGTAGCTAGCTAGCTAGCTAC

# This sequence has a deletion of GCT relative to the one above
TAGCTAGCTAGCTAGCTAGCTGATCA   GATCGTAGCTAGTCGATCGTAGCTAGCTAGCTAGCTAC
```

Genome and transcriptome references are usually stored in the *FASTA* file format. FASTA files usually have a filename extension of ".fasta" or ".fa". FASTA is similar to the FASTQ format discussed in Section 1.5, but without the quality scores. Each entry in a FASTA file has a comment line beginning with '>' (rather then '@' as in FASTQ), followed by one or more lines of sequence data. Note that a genome reference file typically has very few entries (one for each chromosome) and each entry is huge, often millions of bases. The comment line will contain the chromosome name (1, 2, 3, X, Y, etc.) and some additional information about the chromosome such as the source, the length, etc. It is mostly free-format, so not standardized, and may contain just about anything deemed useful by the project that generated the file. A portion of our yeast genome reference is shown below.

```
>1 dna:chromosome chromosome:R64-1-1:1:1:230218:1 REF
CCACACCACACCCACACACCCACACACCACACCACACACCACACCACACCCACACACACA
CATCCTAACACTACCCTAACACAGCCCTAATCTAACCCTGGCCAACCTGTCTCTCAACTT
ACCCTCCATTACCCTGCCTCCACTCGTTACCCTGTCCCATTCAACCATACCACTCCGAAC
CACCATCCATCCCTCTACTTACTACCACTCACCCACCGTTACCCTCCAATTACCCATATC
CAACCCACTGCCACTTACCCTACCATTACCCTACCATCCACCATGACCTACTCACCATAC
TGTTCTTCTACCCACCATATTGAAACGCTAACAAATGATCGTAAATAACACACACGTGCT
TACCCTACCACTTTATACCACCACCACATGCCATACTCACCCTCACTTGTATACTGATTT
TACGTACGCACACGGATGCTACAGTATATACCATCTCAAACTTACCCTACTCTCAGATTC
CACTTCACTCCATGGCCCATCTCTCACTGAATCAGTACCAAATGCACTCACATCATTATG
CACGGCACTTGCCTCAGCGGTCTATACCCTGTGCCATTTACCCATAACGCCCATCATTAT
```

```
CCACATTTTGATATCTATATCTCATTCGGCGGTCCCAAATATTGTATAACTGCCCTTAAT

[ Output omitted for brevity ]

>2 dna:chromosome chromosome:R64-1-1:2:1:813184:1 REF
AAATAGCCCTCATGTACGTCTCCTCCAAGCCCTGTTGTCTCTTACCCGGATGTTCAACCA
AAAGCTACTTACTACCTTTATTTTATGTTTACTTTTTATAGGTTGTCTTTTTATCCCACT
TCTTCGCACTTGTCTCTCGCTACTGCCGTGCAACAAACACTAAATCAAAACAATGAAATA
CTACTACATCAAAACGCATTTTCCCTAGAAAAAAAATTTTCTTACAATATACTATACTAC
ACAATACATAATCACTGACTTTCGTAACAACAATTTCCTTCACTCTCCAACTTCTCTGCT
CGAATCTCTACATAGTAATATTATATCAAATCTACCGTCTGGAACATCATCGCTATCCAG
CTCTTTGTGAACCGCTACCATCAGCATGTACAGTGGTACCCTCGTGTTATCTGCAGCGAG
AACTTCAACGTTTGCCAAATCAAGCCAATGTGGTAACAACCACATCTCCGAAATCTGCTC
CAAAAGATATTCCAGTTTCTGCCGAAATGTTTTATTGTAGAACAGCCCTATCAGCATCGA
CAGGAATGCCGTCCAATGCGGCACTTTAGATGGGGTAACTCCCAGCGCAAGCTGATCTCG
CAAGTGCATTCCTAGACTTAATTCATATCTGCTCCTCAACTGTCGATGATGCCTGCTAAA
CTGCAGCTTGACGTACTGCGGACCCTGCAGTCCAGCGCTCGTCATGGAACGCAAACGCTG
```

> **Caution**
>
> Sort order of features in FASTA and other files is often important to the software tools that work with them, and often must be the same in input files used by the software (e.g. the genome reference and the feature file, discussed later). Consistent sort order in all input files allows programs to operate efficiently, reading through each file only once, rather than jumping around looking for the data they need. You will encounter cases where one file is sorted lexically ("10" comes before "9" because the character '1' comes before '9' in ISO character sets) and another is sorted numerically ("9" comes before "10"). You will need to bring them into agreement using the appropriate software tools before processing can succeed.

A transcriptome FASTA file uses the same format as a genome reference, but the comment line for each entry specifies the feature (usually a transcript) name and also the location of the feature within the genome. Below is part of the transcriptome for our yeast data. The "mRNA" in the feature names below indicates that they are transcripts, as does "cdna" (complementary DNA derived from an RNA). You might also see names containing "gene", "exon", "snRNA", etc. The third field in this example contains the chromosome number (unfortunately in Roman numerals, "XVI" = 16), and offset. The fourth field indicates the *parent* feature of which the transcript is part. The overall format of the comment varies greatly, but fortunately you won't need to worry about it for a simple differential expression analysis.

```
>YPL071C_mRNA cdna chromosome:R64-1-1:XVI:420048:420518:-1 gene:YPL071C
ATGAGTTCCCGGTTTGCAAGAAGTAATGGCAATCCCAACCACATTAGGAAAAGAAATCAT
TCTCCAGACCCAATAGGAATTGATAATTATAAAAGAAAAAGACTAATTATAGATTTAGAG
AATTTATCCTTAAATGATAAAGGGCCCAAGAACGGACATGCAGATGATAACAATCTTATT
CATAACAATATAGTATTCACAGACGCTATTGATGATAAGGTCCTGAAAGAGATCATCAAG
TGTTCCACAAGTAAACGCGGCGACAATGACTTGTTTTATGACAAAATATGGGAACGTTTG
AGAGAAAAAAGGCTACAAATAATAAAATGGGTAGATTATAAGGAAATTGCTTATCTAAGC
TGGTGGAAGTGGTTCCATAATCAAATGACTTCGAAATACACTTATGATGGAGAGGCTGAT
ACCGATGTTGAAATGATGGCAGTGGATACTGATGTGGATATGGATGCGTAA
>YLL050C_mRNA cdna chromosome:R64-1-1:XII:39804:40414:-1 gene:YLL050C
ATGTCTAGATCTGGTGTTGCTGTTGCTGATGAATCCCTTACCGCTTTCAATGACTTGAAA
TTGGGTAAAAAATACAAATTTATTTTATTCGGATTGAACGATGCTAAAACCGAAATCGTT
GTCAAGGAAACCTCTACTGACCCATCTTACGATGCCTTCTTAGAGAAATTGCCAGAAAAC
GACTGTCTTTACGCCATTTACGATTTTGAATACGAAATTAATGGTAATGAAGGTAAGAGA
TCCAAGATTGTTTTCTTCACTTGGTCTCCAGACACTGCTCCAGTCAGATCTAAGATGGTC
TATGCATCCTCCAAGGATGCCTTAAGAAGAGCCTTAAACGGTGTCTCTACCGATGTTCAA
GGTACTGATTTTTCCGAAGTTTCTTACGATTCTGTTTTGGAAAGAGTCAGCAGAGGCGCT
GGTTCTCATTAA
```

The completeness of reference genomes and transcriptomes varies widely. At the time of this writing, there are relatively few model organisms for which mature references exist. Among the most mature are Arabidopsis, fruit fly, human, mouse, and yeast. Building a reference genome is an expensive project requiring vast computing resources. It involves purifying and fragmenting

DNA, sequencing as many fragments as possible, and then using software such as Canu to *assemble* the genome. Genome assemblers look for overlaps in the fragments that most likely indicate they are from the same location in the genome. The results are not always clear, due in part to the presence of long, repetitive DNA sequences in most genomes. If there is a repetitive region 1,000 bases long, and our reads are 150 bases long, there is simply no way to determine the actual length of the repetitive region is using the reads alone. Newer *long read* sequencing technology that can sequence fragments many thousands of bases in length are making genome assembly easier, but the origin of some fragments might still be impossible to determine with certainty. It usually takes many iterations of genome assembly and information from other sources to build a fairly complete reference.

Most reference genomes are available from the three major global repositories, namely the U.S.A.-based, Genbank, European-based Ensembl, and the DNA data bank of Japan. These three sites synchronize with each other regularly, so most sequence data can be retrieved from any of the three. The choice will usually be a matter of geographic distance (downloading from closer sites is faster) or which website interface you find preferable. Additional sites specific to a certain model organisms, such as the Zebrafish Information Network, may have more recent genome references and other information available. Use of these sites is ever-changing and not the focus of this text, so we don't delve into details here. For more information, consult a bioinformatics overview text such as The Biostar Handbook.

4.10.2 The genome reference

If we want to map our reads to an entire genome, simply downloading a reference FASTA file might suffice. Often, however, we want to exclude X and Y chromosomes, or unassembled sequences from less mature reference genomes. In these cases, we can download references for individual chromosomes and concatenate them, or download the entire genome and remove the chromosomes or unassembled sequences we don't want.

You can, of course, download the reference files you need using a web browser. However, to make the process faster, fully documented, and more reproducible, it is better to use a command-line fetch tool from a shell script. Writing scripts for each stage fully documents the analysis and makes it trivial to rerun any stage. The most popular command-line fetch tool at the time of this writing is **curl**, which is part of the default install of many Unix-compatible systems, and easily installed using a package manager on the rest. All you need is the URL (uniform resource locator) of the file, which is shown by a web browser.

Doing a simple web search for "ensembl yeast" will lead us to the page shown in Figure 4.20.

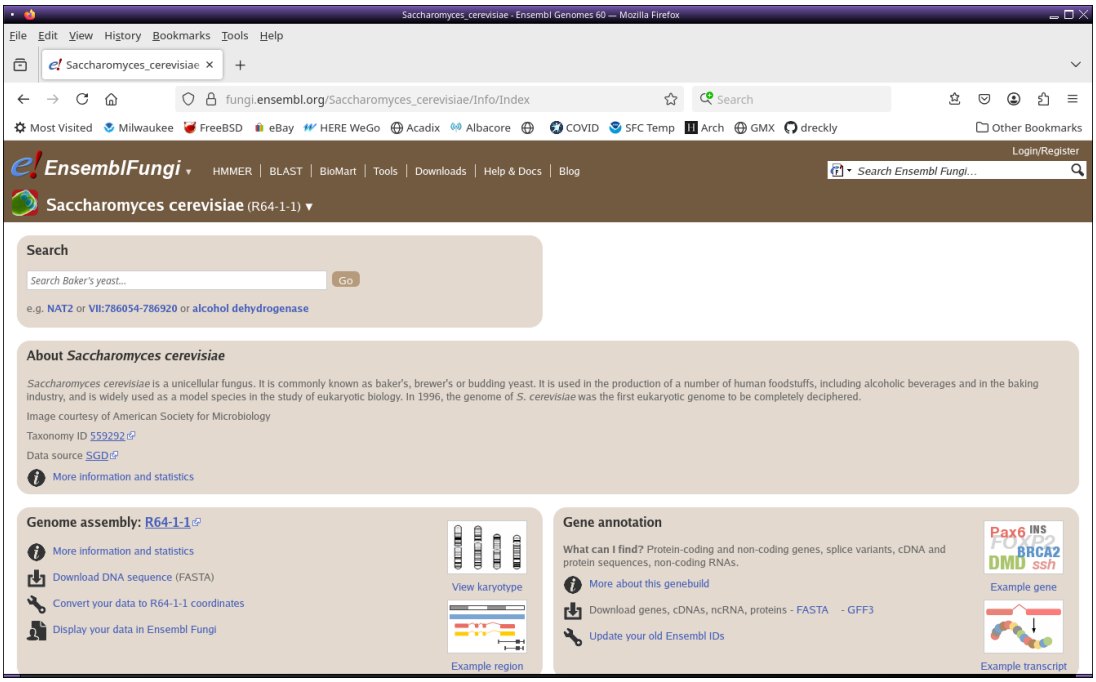

Figure 4.20: Ensembl yeast web page

Clicking on "Download DNA sequence (FASTA)" should lead us to the file listing shown in Figure 4.21, where we can right-click on any filename and select "Copy link" from the popup menu. The "Copy link" will save the complete URL of the file to your desktop clipboard, so it can be pasted into another application. The ".dna.toplevel" file is the generic complete genome reference. See the README file in the same folder for a description of the files.

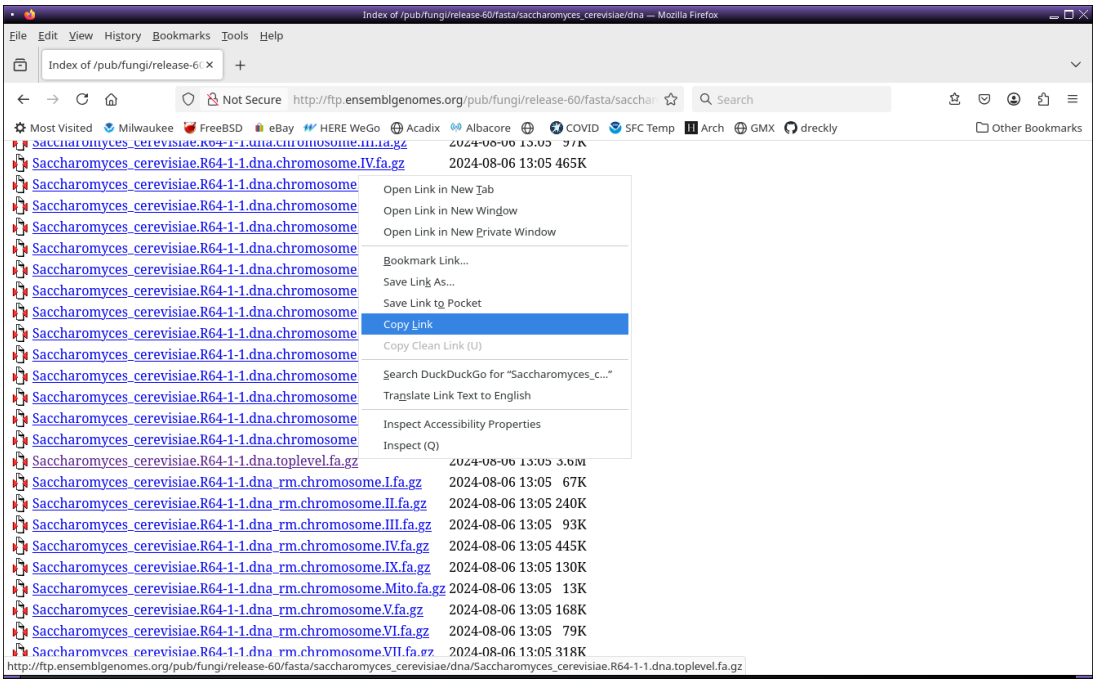

Figure 4.21: Copy link to whole genome reference

We can then paste this URL into a script opened in a text editor as shown in Figure 4.22.

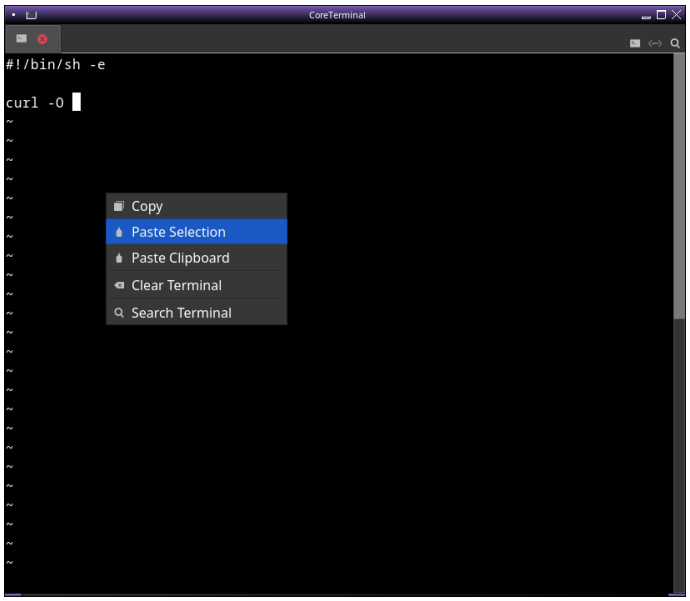

Figure 4.22: Pasting the URL for the yeast genome reference

After pasting the URL, the script should appear as shown in Figure 4.23.

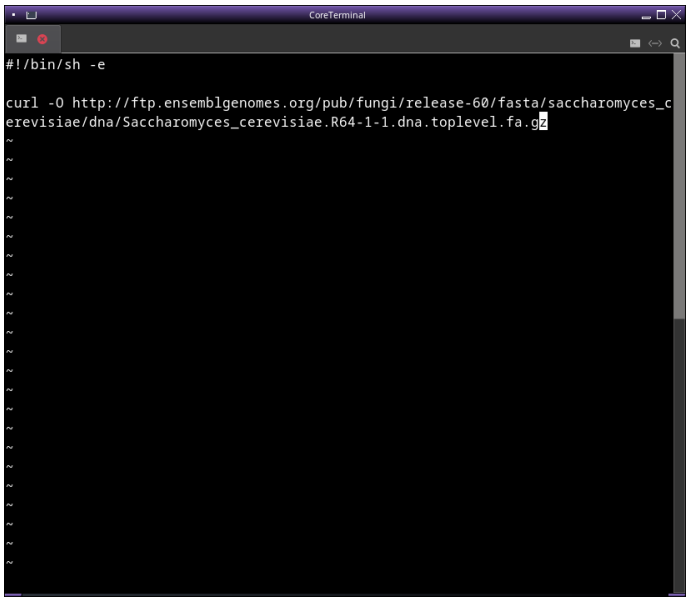

Figure 4.23: Our script after pasting the URL

We can then run the script to download the reference FASTA as follows:

```
# Make the script executable if you haven't already
shell-prompt: chmod a+x get-reference.sh

# Run the script
shell-prompt: ./get-reference.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 3695k  100 3695k    0     0  2898k      0  0:00:01  0:00:01 --:--:-- 2900k
```

Before moving on, we want to verify that the reference file is intact. In line with the "know your data" principle, we will want to first have a look at the content:

```
# Have a quick look at the file.
# "zmore" is a convenience command the combines "gunzip -c" and "more".
shell-prompt: zmore Saccharomyces_cerevisiae.R64-1-1.dna.toplevel.fa.gz
>I dna:chromosome chromosome:R64-1-1:I:1:230218:1 REF
CCACACCACACCCACACACCCACACACCACACCACACACCACACCACACCCACACACACA
CATCCTAACACTACCCTAACACAGCCCTAATCTAACCCTGGCCAACCTGTCTCTCAACTT
ACCCTCCATTACCCTGCCTCCACTCGTTACCCTGTCCCATTCAACCATACCACTCCGAAC
CACCATCCATCCCTCTACTTACTACCACTCACCCACCGTTACCCTCCAATTACCCATATC
CAACCCACTGCCACTTACCCTACCATTACCCTACCATCCACCATGACCTACTCACCATAC
TGTTCTTCTACCCACCATATTGAAACGCTAACAAATGATCGTAAATAACACACACGTGCT
TACCCTACCACTTTATACCACCACCACATGCCATACTCACCCTCACTTGTATACTGATTT
TACGTACGCACACGGATGCTACAGTATATACCATCTCAAACTTACCCTACTCTCAGATTC
CACTTCACTCCATGGCCCATCTCTCACTGAATCAGTACCAAATGCACTCACATCATTATG
CACGGCACTTGCCTCAGCGGTCTATACCCTGTGCCATTTACCCATAACGCCCATCATTAT

[ Output omitted for brevity ]
```

```
# Same as above:
shell-prompt: gunzip -c Saccharomyces_cerevisiae.R64-1-1.dna.toplevel.fa.gz | more
```

We can then do a quick check to ensure that all of the chromosomes we want, and only the chromosomes we want, are present in the file:

```
# See what chromosomes are in the reference FASTA.
# '^>' is a grep pattern with '^' meaning the beginning of the line
# so show lines with '>' as the first character (FASTA comments).
# "zgrep" is a convenience command that combines "gunzip -c" and "grep".
shell-prompt: zgrep '^>' Saccharomyces_cerevisiae.R64-1-1.dna.toplevel.fa.gz
>I dna:chromosome chromosome:R64-1-1:I:1:230218:1 REF
>II dna:chromosome chromosome:R64-1-1:II:1:813184:1 REF
>III dna:chromosome chromosome:R64-1-1:III:1:316620:1 REF
>IV dna:chromosome chromosome:R64-1-1:IV:1:1531933:1 REF
>V dna:chromosome chromosome:R64-1-1:V:1:576874:1 REF
>VI dna:chromosome chromosome:R64-1-1:VI:1:270161:1 REF
>VII dna:chromosome chromosome:R64-1-1:VII:1:1090940:1 REF
>VIII dna:chromosome chromosome:R64-1-1:VIII:1:562643:1 REF
>IX dna:chromosome chromosome:R64-1-1:IX:1:439888:1 REF
>X dna:chromosome chromosome:R64-1-1:X:1:745751:1 REF
>XI dna:chromosome chromosome:R64-1-1:XI:1:666816:1 REF
>XII dna:chromosome chromosome:R64-1-1:XII:1:1078177:1 REF
>XIII dna:chromosome chromosome:R64-1-1:XIII:1:924431:1 REF
>XIV dna:chromosome chromosome:R64-1-1:XIV:1:784333:1 REF
>XV dna:chromosome chromosome:R64-1-1:XV:1:1091291:1 REF
>XVI dna:chromosome chromosome:R64-1-1:XVI:1:948066:1 REF
>Mito dna:chromosome chromosome:R64-1-1:Mito:1:85779:1 REF
```

```
# Same as above:
shell-prompt: gunzip -c Saccharomyces_cerevisiae.R64-1-1.dna.toplevel.fa.gz | grep '^>'
```

To be absolutely sure that the file was downloaded correctly, we can verify the checksum data. (Well, almost absolutely: it is theoretically possible for two different files to have the same checksum, but astronomically unlikely.) In the same directory as the file at http://ftp.ensemblgenomes.org/pub/fungi/release-60/fasta/saccharomyces_cerevisiae/dna/, there is a file called `CHECKSUMS`. We can either left-click on it to view it in the web browser, or right-click on it and save it to our computer. After downloading, use **grep** to show the line containing our reference filename:

```
shell-prompt: grep Saccharomyces_cerevisiae.R64-1-1.dna.toplevel.fa.gz CHECKSUMS
43929  3696 [ path removed for brevity ] Saccharomyces_cerevisiae.R64-1-1.dna.toplevel.fa. ↩
    gz
```

Though it is (annoyingly) not documented on this web page (it should be in the README file or the CHECKSUMS file, but isn't), the numbers in the CHECKSUMS file are computed using the Unix **sum** command. A web search for "ensembl checksums" eventually pointed me to a conversation with this information. We can run **sum** on our downloaded file and check to see that the numbers match what is in the CHECKSUMS file on the Ensembl site:

```
shell-prompt: sum Saccharomyces_cerevisiae.R64-1-1.dna.toplevel.fa.gz
43929 3696 Saccharomyces_cerevisiae.R64-1-1.dna.toplevel.fa.gz
```

Caution This does not verify that we downloaded the right file, but only that the download completed successfully. It's easy to read too much into the thought "I checked it".

The above reference includes mitochondrial sequences, listed as another chromosome in the FASTA file. We could remove it, or just not download that part to begin with. FASTA files can be easily built by concatenating smaller FASTAs containing individual chromosomes. Below is a script that downloads individual chromosome FASTA files for everything except the mitochondria, and concatenates them into a complete reference.

```
#!/bin/sh -e

# All of the URL except the filename
```

```
site=http://ftp.ensemblgenomes.org/pub/fungi/release-60/fasta/saccharomyces_cerevisiae/dna

# Start with an empty reference file
reference=all-but-mito-genome.fa
rm -f $reference

# Download all desired chromosomes and add them to our reference
for chrom in I II III IV V VI VII VIII IX X XI XII XIII XIV XV XVI; do
    file=Saccharomyces_cerevisiae.R64-1-1.dna.chromosome.$chrom.fa.gz

    # --continue-at - tells curl to resume a previous download
    # This allows us to rerun the script to resume the reference build
    # if it fails.
    curl -O --continue-at - $site/$file

    # Append raw (uncompressed) FASTA to reference
    # With gzip, we could also just concatenate the compressed files,
    # but this feels cleaner
    gunzip -c $file >> $reference
done
# Recompress the entire reference.  Feel free to use a different compression
# tool if you like, assuming the tools used in subsequent stages support it.
gzip $reference
```

4.10.3 The transcriptome reference

There are basically two ways to obtain a transcriptome reference:

- There is a cDNA reference available for download for most organisms. As the name implies, this contains complementary DNA sequences derived from sequenced RNAs.
- A transcriptome can be built using a genome reference and a *feature file*, which describes known features in the genome, such as genes, transcripts, exons, snRNAs, etc. The content of a feature file is often referred to as *annotations*. The feature files generally come in *gene transfer format* (GTF), or *general feature format v3* (GFF3), the successor to GTF.

The features found in the cDNA reference may not match those in the GTF and GFF3 files exactly. The advantage of building from a genome and GTF of GFF3 is that subsequent analysis stages that use the GTF or GFF3 file to look up features will always find the features contained in our reference file.

The latest standard for feature files at the time of this writing is *GFF3* (General Feature Format version 3). *GTF* (Gene Transfer Format) is essentially GFF version 2, and is still widely used, as not all programs support GFF3 yet. GFF3 is a tab-separated file, whose fields are well-defined (https://en.wikipedia.org/wiki/General_feature_format). A GFF3 file is sorted hierarchically, so that features appear directly after their parent feature. I.e., an mRNA (transcript) appears below the gene from which is is transcribed, an exon appears below the transcript of which it is part, and a coding sequence appears below the exon of which it is part, as shown in Figure 4.24.

```
###
1       sgd     gene    1807    2169    .       -       .       ID=gene:YAL068C;Name=PAU8; ↩
    biotype=protein_coding;description=Protein of unknown function%3B member of the  ↩
    seripauperin multigene family encoded mainly in subtelomeric regions [Source:SGD%3BAcc: ↩
    S000002142];gene_id=YAL068C;logic_name=sgd
1       sgd     mRNA    1807    2169    .       -       .       ID=transcript:YAL068C_mRNA; ↩
    Parent=gene:YAL068C;Name=PAU8;biotype=protein_coding;transcript_id=YAL068C_mRNA
1       sgd     exon    1807    2169    .       -       .       Parent=transcript: ↩
    YAL068C_mRNA;Name=YAL068C_mRNA-E1;constitutive=1;ensembl_end_phase=0;ensembl_phase=0; ↩
    exon_id=YAL068C_mRNA-E1;rank=1
1       sgd     CDS     1807    2169    .       -       0       ID=CDS:YAL068C;Parent= ↩
    transcript:YAL068C_mRNA;protein_id=YAL068C
###
1       sgd     gene    2480    2707    .       +       .       ID=gene:YAL067W-A;biotype= ↩
    protein_coding;description=Putative protein of unknown function%3B identified by gene- ↩
    trapping%2C microarray-based expression analysis%2C and genome-wide homology searching [ ↩
    Source:SGD%3BAcc:S000028593];gene_id=YAL067W-A;logic_name=sgd
1       sgd     mRNA    2480    2707    .       +       .       ID=transcript:YAL067W- ↩
    A_mRNA;Parent=gene:YAL067W-A;biotype=protein_coding;transcript_id=YAL067W-A_mRNA
1       sgd     exon    2480    2707    .       +       .       Parent=transcript:YAL067W- ↩
    A_mRNA;Name=YAL067W-A_mRNA-E1;constitutive=1;ensembl_end_phase=0;ensembl_phase=0;exon_id ↩
    =YAL067W-A_mRNA-E1;rank=1
1       sgd     CDS     2480    2707    .       +       0       ID=CDS:YAL067W-A;Parent= ↩
    transcript:YAL067W-A_mRNA;protein_id=YAL067W-A
###
```

Figure 4.24: Excerpt from a GFF3 file

For an example of downloading a cDNA reference, we can revisit the Ensembl yeast web page shown in Figure 4.25.

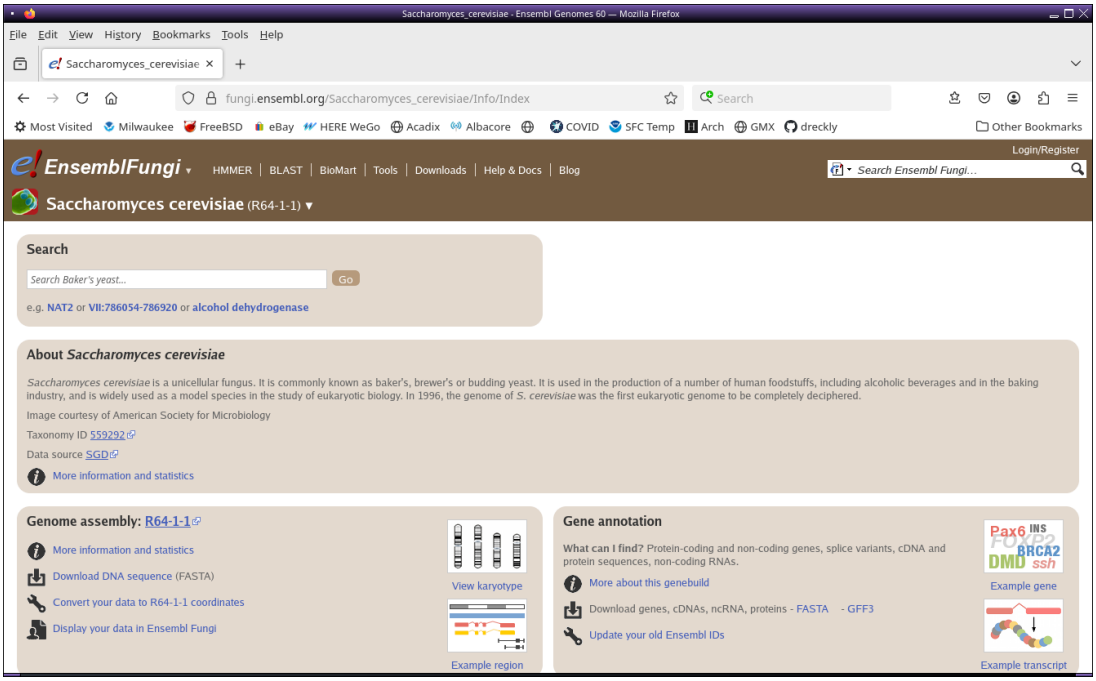

Figure 4.25: Ensembl yeast web page

This time, click on "Download genes, cDNAs, proteins - FASTA" under "Gene annotation". This should open a directory listing as shown by Figure 4.26.

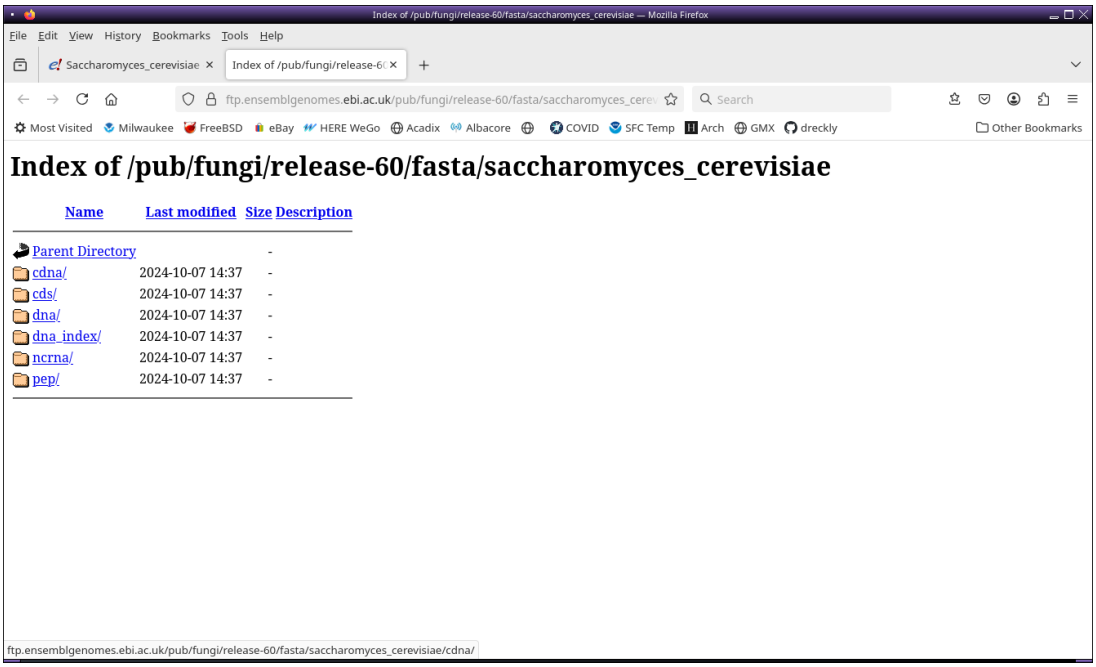

Figure 4.26: Ensembl gene FASTA page

Click on "cDNA", and you should see a file listing as shown in Figure 4.27. Right-click on the ".fa.gz" file and select "Copy Link".

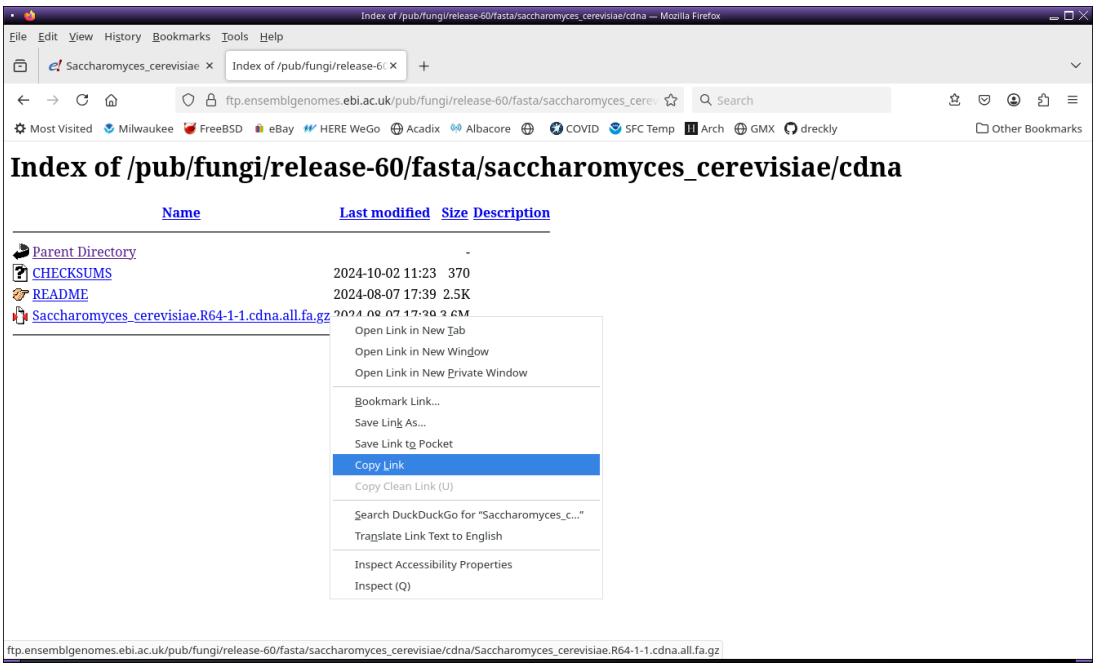

Figure 4.27: Ensembl cDNA page

Then paste the URL into your script as shown in Figure 4.28 and Figure 4.28. Here we introduce a new **curl** option, `--continue-at -`, which tells **curl** to resume a previous download if it did not complete. This can save us a lot of time for large files, though we must be sure that we're downloading the exact same file as before. Otherwise, we might paste parts of different files together and create a Frankenstein monster.

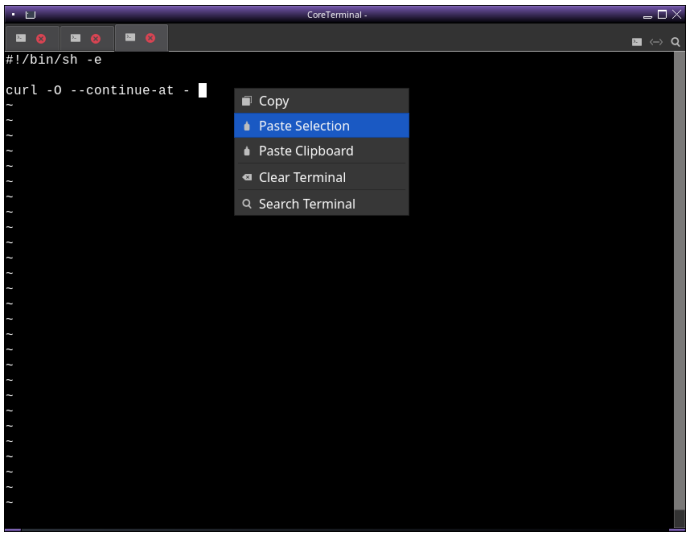

Figure 4.28: Adding the cDNA URL to your script

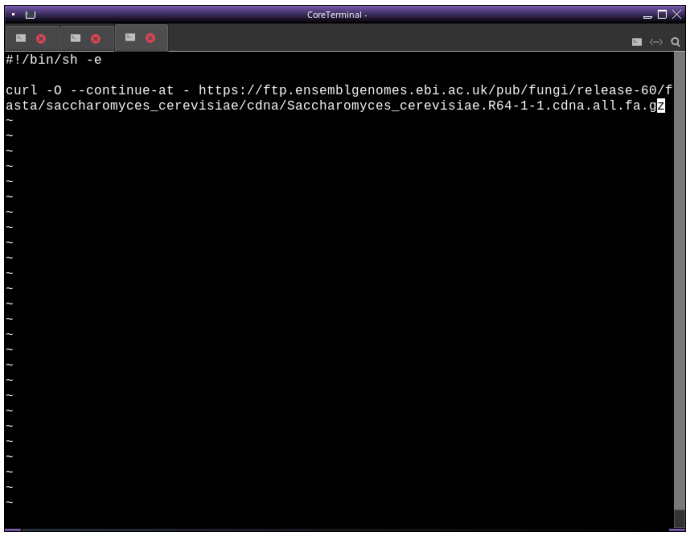

Figure 4.29: The cDNA URL in your script

We can then run the script as often as needed:

```
# Make the script executable if you haven't already
shell-prompt: chmod a+x cdna.sh

# Download the cDNA reference
shell-prompt: ./cdna.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 3717k  100 3717k    0     0  2777k      0  0:00:01  0:00:01 --:--:-- 2778k
```

To build a transcriptome from a genome reference and a feature file, first download the GFF3 file from the Ensemble yeast web page shown in Figure 4.25. This time, click on "GFF3" instead of "FASTA" under "Gene annotation". This should open a file listing as shown in Figure 4.30.

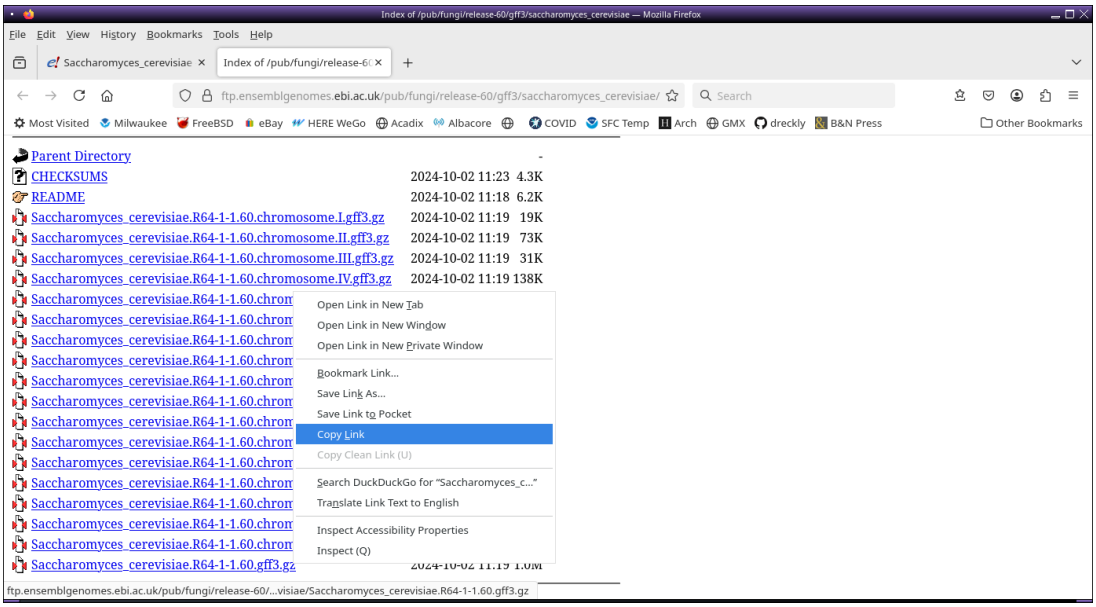

Figure 4.30: Ensembl GFF3 files

Copy and paste the URL of the complete GFF3 file (the one without "chromosome" in its name) into your shell script. We can use the genome reference file we created in Section 4.10.2. We use the **gffread** command, which is installed by the rna-seq meta-package, to build the transcriptome reference. The final script should look something like this:

```
#!/bin/sh -e

# Run the script shown above to build the genome from individual
# chromosome FASTAs.
./build-genome.sh

# Here we split the URL of the GFF3 file into site and GFF
site=https://ftp.ensemblgenomes.ebi.ac.uk/pub/fungi/release-60/gff3/ ↩
    saccharomyces_cerevisiae
gff=Saccharomyces_cerevisiae.R64-1-1.60.gff3.gz

curl -O --continue-at - $site/$gff

# Build reference transcriptome from genome and GTF
transcriptome=all-but-mito-transcriptome.fa

# gffread is installed as part of the rna-seq meta-package
gffread -w $transcriptome -g all-but-mito-genome.fa $gff
```

The **08-reference.sh** script in the DIY-RNA-Seq-DE repository automatically assembles a genome reference from individual chromosomes downloaded separately, assembles a GFF3 file from the individual chromosome GFFs, and downloads a cDNA transcriptome.

```
shell-prompt: ./08-reference.sh

[ Output omitted for brevity ]

Fetching Saccharomyces_cerevisiae.R64-1-1.106.chromosome.XVI.gff3.gz...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 84515  100 84515    0     0   200k      0 --:--:-- --:--:-- --:--:--  200k
Generating chromosome-sizes.tsv...
1       230218
```

```
2       813184
3       316620
4       1531933
5       576874
6       270161
7       1090940
8       562643
9       439888
10      745751
11      666816
12      1078177
13      924431
14      784333
15      1091291
16      948066
```

```
# To save terminal output to a file:
shell-prompt: ./08-reference.sh |& tee reference.out

# If the syntax above doesn't work in your shell:
shell-prompt: ./08-reference.sh 2>&1 | tee reference.out
```

Note

Be sure to carefully review the output to the terminal for warnings, errors, and reasonable values. If you saved a copy of the terminal output by piping through **tee**, you can use tools such as **grep -i error file.out** to search for error messages, etc.

As a quick sanity check for the transcriptome reference, we can count the features and compare them to the transcript count in the GFF3 feature file. The counts should be very similar, though not necessarily identical.

```
# Count features in the cDNA transcriptome reference
shell-prompt: grep '^>' Results/08-reference/all-but-mito-transcriptome.fa | wc -l
    6584

# Count features in the GFF3-derived transcriptome reference
shell-prompt: grep '^>' Results/08-reference/all-but-mito-transcriptome-from-gff.fa | wc -l
    6981

# Count features (transcripts) in the GFF3
shell-prompt: awk '$3 == "mRNA"' Results/08-reference/Saccharomyces_cerevisiae.R64-1-1.106. ↩
    gff3 | wc -l
    6572
```

As always, we should know our data by looking over the files.

```
shell-prompt: head Results/08-reference/*.fa
==> Results/08-reference/all-but-mito-genome.fa <==
>1 dna:chromosome chromosome:R64-1-1:1:1:230218:1 REF
CCACACCACACCCACACACCCACACACCACACCACACACCACACCACACCCACACACACA
CATCCTAACACTACCCTAACACAGCCCTAATCTAACCCTGGCCAACCTGTCTCTCAACTT
ACCCTCCATTACCCTGCCTCCACTCGTTACCCTGTCCCATTCAACCATACCACTCCGAAC
CACCATCCATCCCTCTACTTACTACCACTCACCCACCGTTACCCTCCAATTACCCATATC
CAACCCACTGCCACTTACCCTACCATTACCCTACCATCCACCATGACCTACTCACCATAC
TGTTCTTCTACCCACCATATTGAAACGCTAACAAATGATCGTAAATAACACACACGTGCT
TACCCTACCACTTTATACCACCACCACATGCCATACTCACCCTCACTTGTATACTGATTT
TACGTACGCACACGGATGCTACAGTATATACCATCTCAAACTTACCCTACTCTCAGATTC
CACTTCACTCCATGGCCCATCTCTCACTGAATCAGTACCAAATGCACTCACATCATTATG

==> Results/08-reference/all-but-mito-transcriptome-from-gff.fa <==
```

```
>transcript:YAL069W_mRNA CDS=1-315
ATGATCGTAAATAACACACACGTGCTTACCCTACCACTTTATACCACCACCACATGCCATACTCACCCTC
ACTTGTATACTGATTTTACGTACGCACACGGATGCTACAGTATATACCATCTCAAACTTACCCTACTCTC
AGATTCCACTTCACTCCATGGCCCATCTCTCACTGAATCAGTACCAAATGCACTCACATCATTATGCACG
GCACTTGCCTCAGCGGTCTATACCCTGTGCCATTTACCCATAACGCCCATCATTATCCACATTTTGATAT
CTATATCTCATTCGGCGGTCCCAAATATTGTATAA
>transcript:YAL068W-A_mRNA CDS=1-255
ATGCACGGCACTTGCCTCAGCGGTCTATACCCTGTGCCATTTACCCATAACGCCCATCATTATCCACATT
TTGATATCTATATCTCATTCGGCGGTCCCAAATATTGTATAACTGCCCTTAATACATACGTTATACCACT
TTTGCACCATATACTTACCACTCCATTTATATACACTTATGTCAATATTACAGAAAAATCCCCACAAAAA

==> Results/08-reference/all-but-mito-transcriptome.fa <==
>YPL071C_mRNA cdna chromosome:R64-1-1:XVI:420048:420518:-1 gene:YPL071C gene_biotype: ↩
    protein_coding transcript_biotype:protein_coding description:Putative protein of unknown ↩
     function; green fluorescent protein (GFP)-fusion protein localizes to both the  ↩
    cytoplasm and the nucleus [Source:SGD;Acc:S000005992]
ATGAGTTCCCGGTTTGCAAGAAGTAATGGCAATCCCAACCACATTAGGAAAAGAAATCAT
TCTCCAGACCCAATAGGAATTGATAATTATAAAAGAAAAAGACTAATTATAGATTTAGAG
AATTTATCCTTAAATGATAAAGGGCCCAAGAACGGACATGCAGATGATAACAATCTTATT
CATAACAATATAGTATTCACAGACGCTATTGATGATAAGGTCCTGAAAGAGATCATCAAG
TGTTCCACAAGTAAACGCGGCGACAATGACTTGTTTTATGACAAAATATGGGAACGTTTG
AGAGAAAAAAGGCTACAAATAATAAAATGGGTAGATTATAAGGAAATTGCTTATCTAAGC
TGGTGGAAGTGGTTCCATAATCAAATGACTTCGAAATACACTTATGATGGAGAGGCTGAT
ACCGATGTTGAAATGATGGCAGTGGATACTGATGTGGATATGGATGCGTAA
>YLL050C_mRNA cdna chromosome:R64-1-1:XII:39804:40414:-1 gene:YLL050C gene_biotype: ↩
    protein_coding transcript_biotype:protein_coding gene_symbol:COF1 description:Cofilin,  ↩
    involved in pH-dependent actin filament depolarization; binds both actin monomers and  ↩
    filaments and severs filaments; involved in the selective sorting, export of the  ↩
    secretory cargo from the late golgi; genetically interacts with pmr1; thought to be  ↩
    regulated by phosphorylation at SER4; ubiquitous and essential in eukaryotes [Source:SGD ↩
    ;Acc:S000003973]

shell-prompt: more Results/08-reference/all-but-mito-genome.fa

[ Output omitted for brevity ]

shell-prompt: more Results/08-reference/all-but-mito-transcriptome-from-gff.fa

[ Output omitted for brevity ]

shell-prompt: more Results/08-reference/all-but-mito-transcriptome.fa

[ Output omitted for brevity ]
```

At this point, you should have your trimmed reads and reference files in-place, so you are ready to perform the read-mapping stage presented in Section 4.11. You can use the same methods discussed in Section 4.10.2 to verify the integrity of your transcriptome FASTAs.

4.10.4 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

---

1. What is a reference genome?
2. What is a reference transcriptome?
3. Why don't we use the sequences from a real individual as a reference?
4. What are the common types of differences in DNA sequences between two individuals?

5. What file format usually contains reference genomes and transcriptomes? How is a sequence represented in a FASTA file?
6. Why is the sort order of reference files important?
7. How accurate are the reference genomes and transcriptomes for most organisms? Why?
8. Where can we find most reference genomes and transcriptomes? Which one should we use?
9. Why would we choose not to simply download a genome reference as one file?
10. What is the advantage of using a command-line download tool such as **curl** instead of downloading reference file interactively with a web browser?
11. Show a **curl** command for downloading chromosome 1 of Mus musculus (common mouse) from Ensemble.
12. Show a command for viewing the content of the mouse chromosome 1 downloaded in the previous question, one screen at a time.
13. What are checksums for? How to they work?
14. What is a feature file?
15. How do we obtain a reference transcriptome?
16. What is a simple sanity check for a transcriptome reference built from a genome and feature file?

4.11 Read mapping (Alignment)

The term *alignment* could actually mean any matching of sequences, but is often used to refer to the *read mapping* stage of a differential expression analysis. Read mapping attempts to align reads from a sequencer to a reference transcriptome or genome. In doing so, we hope to determine the location within the genome or transcriptome from which each fragment in the DNA/RNA library originated. Strictly speaking, when aligning to a transcriptome reference, we're not concerned about the physical position within the genome, but which known feature (gene, transcript, exon, ...) the read came from. The output of most read mappers is a file called an *alignment map*, which lists one or more location within the reference to which each read aligns well, along with an alignment score to indicate how closely the read matches that location. The alignment map is usually in *SAM* (*Sequence Alignment Map*) format, which is a plain text file that you can process with normal Unix commands like **more** and **grep**, or its compressed binary equivalent *BAM* format. You can convert between BAM and SAM using **samtools view**. The samtools software suite is installed as part of the rna-seq meta-package.

For aligning to a transcriptome, we use Kallisto, a very fast *pseudoalignment* tool [Kallisto]. Kallisto is called a pseudoaligner because it does not attempt to compare full sequences to the reference. Instead, it performs only partial alignments and uses statistical methods to determine the best match. This results in a much faster alignment process with results comparable to full aligners.

For mapping to a genome, we mainly use HISAT2, a *splice-aware* aligner which is reliable and efficient [Hisat2]. HISAT2 is the successor of another popular aligner called TopHat.

Another popular open source splice-aware aligner is STAR [STAR], which has much higher memory requirements that may actually prevent it from running on a typical PC for a large and complex genome. One aligner comparison study reported that 96% of the reads were mapped to the exact same location by HISAT2 and STAR [Schaarschmidt-align]. Hence, results of your analysis are not likely to change much depending on which aligner you use. However, if you have access to hardware resources capable of running STAR, it might be worthwhile to run both HISAT2 and STAR, and compare the results for your data. Every data set is different and trying out multiple tools may yield more insights.

Note The benefits of exploring alternative aligners and quantifiers underscores the value of using package managers, as discussed in Chapter 3. The ability to install multiple tools easily, reliably, and securely, makes casual exploration much more feasible, potentially leading to a better understanding of bioinformatics and superior research results.

Read mappers use an *index* file to speed up the alignment process. This index file is typically generated by another tool in the same aligner package prior to doing the alignments. The index, in effect, facilitates very fast searching of the reference genome or transcriptome. The naive alternative to using an index would be searching the entire reference for each of the millions of reads in our FASTQ files. This approach would be extremely slow and wasteful. An index, just like the index in the back of a book, allows the aligner to locate matches in the reference much faster.

4.11.1 Running Kallisto

To use **kallisto**, we first build a Kallisto index file, which in turn utilizes a standard FASTA file index, which we can generate using **samtools**, as shown below. There are many options for generating the Kallisto index, which you can find in the Kallisto documentation. Here we just generate an index with default options.

```
# Generate a standard FASTA index called all-but-mito-genome.fa.fai
shell-prompt: samtools faidx all-but-mito-transcriptome.fa

# Build the specialized kallisto index file
shell-prompt: kallisto index --index=kallisto.index all-but-mito-transcriptome.fa
```

The DIY-RNA-Seq-DE repository contains a script that will run **kallisto index** using the transcriptome references created by **08-reference.sh**.

```
shell-prompt: ./09-kallisto-index.sh

[ Output omitted for brevity ]

[build] loading fasta file Results/08-reference/all-but-mito-transcriptome.fa
[build] k-mer length: 31
KmerStream::KmerStream(): Start computing k-mer cardinality estimations (1/2)
KmerStream::KmerStream(): Start computing k-mer cardinality estimations (1/2)
KmerStream::KmerStream(): Finished
CompactedDBG::build(): Estimated number of k-mers occurring at least once: 8230600
CompactedDBG::build(): Estimated number of minimizer occurring at least once: 2060650
CompactedDBG::filter(): Processed 8548145 k-mers in 6584 reads
CompactedDBG::filter(): Found 8214523 unique k-mers
CompactedDBG::filter(): Number of blocks in Bloom filter is 56265
CompactedDBG::construct(): Extract approximate unitigs (1/2)
CompactedDBG::construct(): Extract approximate unitigs (2/2)
CompactedDBG::construct(): Closed all input files

CompactedDBG::construct(): Splitting unitigs (1/2)

CompactedDBG::construct(): Splitting unitigs (2/2)
CompactedDBG::construct(): Before split: 14120 unitigs
CompactedDBG::construct(): After split (1/1): 14120 unitigs
CompactedDBG::construct(): Unitigs split: 177
CompactedDBG::construct(): Unitigs deleted: 0

CompactedDBG::construct(): Joining unitigs
CompactedDBG::construct(): After join: 10310 unitigs
CompactedDBG::construct(): Joined 3810 unitigs
[build] building MPHF
[build] creating equivalence classes ...
[build] target de Bruijn graph has k-mer length 31 and minimizer length 23
[build] target de Bruijn graph has 10310 contigs and contains 8221663 k-mers

+ ls -l Results/09-kallisto-index
total 5440
-rw-r--r--  1 bacon bacon 5534406 May 29 11:48 transcriptome.index
```

```
# To save terminal output to a file:
```

```
shell-prompt: ./09-kallisto-index.sh |& tee kallisto-index.out

# If the syntax above doesn't work in your shell:
shell-prompt: ./09-kallisto-index.sh 2>&1 | tee kallisto-index.out
```

Note
Be sure to carefully review the output to the terminal for warnings, errors, and reasonable values. If you saved a copy of the terminal output by piping through **tee**, you can use tools such as **grep -i error file.out** to search for error messages, etc.

After carefully checking the terminal output from **kallisto index** for warnings and errors, and verifying that the index file is a reasonable size (comparable to the transcriptome FASTA from which it was generated), we can move on to the next stage. The next stage of running pseudo-alignments and quantification will further validate the index file content.

```
shell-prompt: ls -l Results/08-reference/all-but-mito-transcriptome.fa Results/09-kallisto- ↩
    index/transcriptome.index
-rw-r--r--  1 bacon bacon 11677183 May 11 07:17 Results/08-reference/all-but-mito- ↩
    transcriptome.fa
-rw-r--r--  1 bacon bacon  5534406 May 29 11:48 Results/09-kallisto-index/transcriptome. ↩
    index
```

We can then proceed to perform pseudo-alignments to the transcriptome for each of our read files. Kallisto is among the fastest aligners around, but alignment will still take a long time compared to the other analysis stages for a complex organism such as a vertebrate. For this reason, **kallisto**, like all aligners, is *multi-threaded*, meaning it can use multiple cooperating processes to speed up completion of a single task. In the case of Kallisto, this means using multiple processes to perform pseudo-alignments and quantification for a single sample FASTQ file.

Note Speedup of multi-threaded programs is often an example of diminishing returns, where using N processors will not come close to reducing the run time by a factor of N. However, most aligners show nearly linear speedup for the number of processors we typically have in a PC. I.e., using four processors will speed up alignment by almost a factor of 4.

At the time of this writing, Kallisto cannot read **zstd** compressed files directly. We might try using a standard Unix technique to get around this: Many Unix programs will read from the standard input if given "-" as a filename argument (or stdin:tag as we saw with **fastqc**), but **kallisto** can't do that either. It requires a filename argument on the command line. Unix provides another workaround for piping data into any programs like **kallisto**. This facility is called a *named pipe*. Recall the unnamed pipe we used to run **fastqc**:

```
shell-prompt: zstdcat file.fastq.zst | fastqc stdin:tag
```

The command above sends the standard output of **zstd** through a Unix pipe (which is really just a memory buffer that acts like a file stream) to the standard input of **fastqc**. The **zstd** process puts output into the pipe buffer, and **fastqc** takes it out. We can't do the same with **kallisto**, because it does not support reading from the standard input. It requires a filename argument on the command-line to specify input. The Unix **mkfifo** (make first-in first-out special file) creates a pipe with an actual filename that can be used for this purpose. We can use a named pipe like an ordinary file, but it doesn't permanently store any of the data on disk. It only buffers data until another process takes it out. The commands below show how to use a named pipe with **kallisto**. Note that the **zstdcat** command is run in the *background*, as indicated by the & at the end of the command. The & tells the shell to start the process, but not wait for it to finish before running the next command. Hence, the **kallisto** command starts running immediately after the **zstdcat** command starts, rather than after it finishes.

```
# Create a named pipe
shell-prompt: mkfifo kallisto-pipe

# First run zstdcat, redirecting the output into the named pipe
# The & at the end puts the process in the background, so the shell
# does not wait for it to finish before running the next command.
```

```
shell-prompt: zstdcat sample1.fastq.zst > kallisto-pipe &

# Run kallisto for single-ended reads, estimated fragment length and
# standard deviation of 190 and 10.  Use four processors.
shell-prompt: mkdir sample1.out
shell-prompt: kallisto quant --single --fragment-length=190 --sd=10 \
                --threads=4 --index=all-but-mito-transcriptome.fa \
                --output-dir=sample1.out kallisto-pipe

# Clean up
rm kallisto-pipe
```

Note that we could also use an ordinary file in place of the named pipe, but this would require writing the entire uncompressed FASTA output before starting **kallisto**:

```
# We don't run zstdcat in the background here.  It has to finish before
# we can start kallisto.
shell-prompt: zstdcat sample1.fastq.zst > temp.fasta
shell-prompt: mkdir sample1.out
shell-prompt: kallisto quant --single --fragment-length=190 --sd=10 \
                --threads=4 --index=all-but-mito-transcriptome.fa \
                --output-dir=sample1.out temp.fasta
rm temp.fasta
```

There are three down sides to using a regular file this way, relative to using a pipe:

- The **zstdcat** command must finish running before the **kallisto** command can start, since reading an ordinary file while another process is writing to it leads to unpredictable results. This will be much slower than using a named pipe, which allows the two processes to run mostly at the same time.
- The temporary file will require a lot of disk space, especially given that it is uncompressed FASTA in this case. This will be much less of a problem on a file system with compression, such as the ZFS filesystem used by GhostBSD, but it's still disk space that we can avoid filling.
- Reading and writing disk, even if using an SSD (solid state drive), is much slower than the memory buffer that a pipe represents. This compounds the slowdown caused by running the processes in sequence rather than in simultaneously.

The DIY-RNA-Seq-DE repository contains a script that will run **kallisto** quantification on all of the trimmed read files produced by **05-fastq-trim.sh**. It runs one **kallisto** job at a time, using all available cores for each job.

```
shell-prompt: ./10-kallisto-quant.sh

[ Output omitted for brevity ]

===
Processing Results/05-trim/sample06-cond1-rep03-trimmed.fastq.zst...
===

+ kallisto quant --single --fragment-length=190 --sd=10 --threads=8 --index=Results/09- ↩
    kallisto-index/transcriptome.index --output-dir=Results/10-kallisto-quant/sample06-cond1 ↩
    -rep03-trimmed sample06-cond1-rep03-trimmed.fifo

[quant] fragment length distribution is truncated gaussian with mean = 190, sd = 10
[index] k-mer length: 31
[index] number of targets: 6,584
[index] number of k-mers: 8,221,663
[quant] running in single-end mode
[quant] will process file 1: sample06-cond1-rep03-trimmed.fifo
[progress] 1M reads processed (70.6% mapped)              done
[quant] processed 1,225,424 reads, 864,716 reads pseudoaligned
[   em] quantifying the abundances ... done
```

```
[   em] the Expectation-Maximization algorithm ran for 709 rounds

+ set +x
```

```
# To save terminal output to a file:
shell-prompt: ./10-kallisto-quant.sh |& tee kallisto-quant.out

# If the syntax above doesn't work in your shell:
shell-prompt: ./10-kallisto-quant.sh 2>&1 | tee kallisto-quant.out
```

Note
Be sure to carefully review the output to the terminal for warnings, errors, and reasonable values. If you saved a copy of the terminal output by piping through **tee**, you can use tools such as **grep -i error file.out** to search for error messages, etc.

To run **kallisto** on paired-end data, we simply add the reverse reads file to the end of the command. If using named pipes, we will need to run two decompression processes, both in the background, sending the output to two separate pipes used as inputs for **kallisto**:

```
# Send raw forward and reverse reads into separate pipes
shell-prompt: zstdcat sample1.fastq-R1.zst > kallisto-pipe-R1 &
shell-prompt: zstdcat sample1.fastq-R2.zst > kallisto-pipe-R2 &

# Give kallisto two raw FASTQ inputs for paired-end processing
shell-prompt: kallisto quant --single --fragment-length=190 --sd=10 \
                --threads=4 --index=all-but-mito-transcriptome.fa \
                --output-dir=sample1.out kallisto-pipe-R1 kallisto-pipe-R2
```

To check the quality of the **kallisto** output, we can start with a basic sanity check: just do a long listing of the abundance files and see if they are all a reasonable size. The main output file are in TSV (tab-separated value) format. The output below shows that all of the output files are a little over 200 KB, which is reasonable. Sometimes a job will fail and leave us with an empty output file (size = 0).

```
shell-prompt: ls -l Results/10-kallisto-quant/*/*.tsv
-rw-r--r--  1 bacon bacon 218653 Apr  9 15:53 Results/10-kallisto-quant/sample01-cond1- ↩
    rep01-trimmed/abundance.tsv
-rw-r--r--  1 bacon bacon 222044 Apr  9 15:53 Results/10-kallisto-quant/sample02-cond2- ↩
    rep01-trimmed/abundance.tsv
-rw-r--r--  1 bacon bacon 221141 Apr  9 15:54 Results/10-kallisto-quant/sample03-cond2- ↩
    rep02-trimmed/abundance.tsv
-rw-r--r--  1 bacon bacon 221011 Apr  9 15:54 Results/10-kallisto-quant/sample04-cond2- ↩
    rep03-trimmed/abundance.tsv
-rw-r--r--  1 bacon bacon 220657 Apr  9 15:54 Results/10-kallisto-quant/sample05-cond1- ↩
    rep02-trimmed/abundance.tsv
-rw-r--r--  1 bacon bacon 219605 Apr  9 15:54 Results/10-kallisto-quant/sample06-cond1- ↩
    rep03-trimmed/abundance.tsv
```

The long listing method is a quick first check that can work with any kind of output, as long as we know what to expect in terms of file size. Another check that's helpful for Kallisto output is counting the lines in the file. The Unix **wc** (word-count) command can count lines, words, and characters in a file. For Kallisto, all of the files should have exactly the same number of lines, which equals the number of features of interest (transcripts in this case) in the reference. All the line counts below match, so we know that none of the **kallisto** processes died prematurely.

```
shell-prompt: wc -l Results/10-kallisto-quant/sample0*/*.tsv
    6585 Results/10-kallisto-quant/sample01-cond1-rep01-trimmed/abundance.tsv
    6585 Results/10-kallisto-quant/sample02-cond2-rep01-trimmed/abundance.tsv
    6585 Results/10-kallisto-quant/sample03-cond2-rep02-trimmed/abundance.tsv
    6585 Results/10-kallisto-quant/sample04-cond2-rep03-trimmed/abundance.tsv
    6585 Results/10-kallisto-quant/sample05-cond1-rep02-trimmed/abundance.tsv
```

```
    6585 Results/10-kallisto-quant/sample06-cond1-rep03-trimmed/abundance.tsv
   39510 total
```

As always, we follow the "know your data" principle and have a look at the results to see if everything looks reasonable. Do the target IDs make sense (they should be mRNAs if we are doing a transcript-based analysis). Do the lengths and counts look reasonable?

```
shell-prompt: head Results/10-kallisto-quant/sample0*/*.tsv
==> Results/10-kallisto-quant/sample01-cond1-rep01-trimmed/abundance.tsv <==
target_id       length  eff_length      est_counts      tpm
YPL071C_mRNA    471     282     14      35.6591
YLL050C_mRNA    432     243     238     703.496
YMR172W_mRNA    2160    1971    42.6436 15.5403
YOR185C_mRNA    663     474     44      66.6754
YLL032C_mRNA    2478    2289    14      4.39312
YBR225W_mRNA    2703    2514    34      9.71415
YEL041W_mRNA    1488    1299    10      5.52945
YOR237W_mRNA    1305    1116    9       5.79255
YMR027W_mRNA    1413    1224    139     81.5689

[ Output omitted for brevity ]
```

Verifying the correctness of the abundances in these files is much more challenging. One approach would be to compare them to abundances computed by other quantification tools such as stringtie and HT-Seq. This would involve running alignments using another tool to produce SAM or BAM alignment maps, and then using the quantifier to produce estimates.

Often, researchers don't make any attempt to verify results at this stage beyond basic sanity checks. Instead, they proceed onto the next stage of the analysis (computing fold-changes) and use the results of that to infer that the alignments and quantification are good.

4.11.2 Running HISAT2

In many cases, we will want to map our reads to a genome in addition to a transcriptome. Besides simply corroborating results from two different methods, another reason for this is the potential for new gene discovery. A transcriptome represents *known* genes, so if there are reads that do not map to any feature in the transcriptome, they simply remain a mystery. Mapping them to the genome is the only way to discover the origin of mRNA reads that do not align to a known gene. At the time of this writing, the biology community is in the very early stages of identifying genes and other features for most organisms. Transcriptomes for all but the most heavily studied model organisms (mouse, human, fruit fly, etc.) are incomplete. Existing *genome* references for many organisms are not that mature either, but there is a chance that they contain coding sequences in the correct locations that are not yet found in the transcriptome.

To map eukaryotic mRNA reads to a genome, there is an additional problem: Most mRNAs are collected from the cytoplasm, after being processed and exported from the nucleus. Most importantly, introns have been removed, so that the mRNA sequences now have large deletions relative to the genome reference. To contend with this issue, we need a *splice-aware* aligner, i.e. one that can recognize a match between a processed mRNA sequence and the full DNA sequence from which it was transcribed, as shown in Figure 4.31.

```
Spliced RNA sequence:   AAUUGUGGUAGCGAGUCAGCUAUCA

                                |---------- Intron ----------|
DNA sequence:   AATTGTGGTAGCGAGTCAGTCGATCGATCGTAGCTAGCTAGCTAGCTAGCTATCA
RNA aligned:    AAUUGUGGUAGCGAGUC------------------------------AGCUAUCA
                                |-- Deletion vs. reference --|
```

Figure 4.31: Aligning a processed mRNA sequence

The two most commonly used splice-aware aligners at the time of this writing are HISAT2 [Hisat2] and STAR [STAR]. These two aligners produce very similar results [Schaarschmidt-align], while STAR has extremely high memory requirements that will likely prevent it from running on a typical PC when analyzing a large genome. We found that 128 GiB of RAM was not enough in some cases, while typical PCs may only have 8 or 16 GiB. For the sake of readers with limited computing resources, we focus on HISAT2 here.

The first step of a HISAT2 alignment, as with most aligners, is to generate an index. The HISAT2 index consists of several ".ht2" files, derived from the genome FASTA file and its generic FASTA index (.fai file). Both of these files should have been downloaded or created as you went though Section 4.10. The HISAT2 index files should be created in the same directory as the FASTA reference file. For our example here, we keep things organized into subdirectories for each pipeline stage. Hence, we link (not copy, which would waste time and disk space) the reference genome and standard FASTA index (.fai file) into the HISAT2 index directory, and then generate the HISAT2 index there:

```
# Make sure we have a standard FASTA index
shell-prompt: samtools faidx Results/08-reference/all-but-mito-genome.fa

# Create HISAT2 index directory
shell-prompt: mkdir Results/12-hisat2-index
shell-prompt: cd Results/12-hisat2-index

# The genome reference and generic index should be in the same directory
# as the HISAT2 index files.  In the unlikely event that 08-reference
# and 12-hisat2 are on different filesystems, use "ln -s" (symbolic links).
shell-prompt: ln ../08-reference/all-but-mito-genome.fa .
shell-prompt: ln ../08-reference/all-but-mito-genome.fa.fai .

# Build HISAT2 index files all-but-mito-genome.fa.1, ...
# The first argument is the FASTA file, the second is the "base" of
# the HISAT2 index filenames, which is often the same.
shell-prompt: hisat2-build all-but-mito-genome.fa all-but-mito-genome.fa
shell-prompt: ls -l
total 34448
-rw-r--r--  1 bacon bacon  8219399 Mar 27 12:06 all-but-mito-genome.1.ht2
-rw-r--r--  1 bacon bacon  3017836 Mar 27 12:06 all-but-mito-genome.2.ht2
-rw-r--r--  1 bacon bacon      152 Mar 27 12:06 all-but-mito-genome.3.ht2
-rw-r--r--  1 bacon bacon  3017832 Mar 27 12:06 all-but-mito-genome.4.ht2
-rw-r--r--  1 bacon bacon  5357645 Mar 27 12:06 all-but-mito-genome.5.ht2
-rw-r--r--  1 bacon bacon  3071004 Mar 27 12:06 all-but-mito-genome.6.ht2
-rw-r--r--  1 bacon bacon       12 Mar 27 12:06 all-but-mito-genome.7.ht2
-rw-r--r--  1 bacon bacon        8 Mar 27 12:06 all-but-mito-genome.8.ht2
-rw-r--r--  2 bacon bacon 12273406 Mar 27 12:04 all-but-mito-genome.fa
-rw-r--r--  2 bacon bacon      375 Mar 27 12:04 all-but-mito-genome.fa.fai
```

Note Some of the HISAT2 index files will likely be very small. This is normal, so don't be alarmed that the index build may have failed.

The DIY-RNA-Seq-DE repository contains a script to automate the commands above:

```
shell-prompt: ./12-hisat2-index.sh

[ Output omitted for brevity ]

Wrote 5357645 bytes to primary GFM file: Results/12-hisat2-index/all-but-mito-genome.fa.5. ↩
    ht2
Wrote 3071004 bytes to secondary GFM file: Results/12-hisat2-index/all-but-mito-genome.fa ↩
    .6.ht2
Re-opening _in5 and _in5 as input streams
Returning from HGFM constructor
Headers:
```

```
    len: 12071326
    gbwtLen: 12071327
    nodes: 12071327
    sz: 3017832
    gbwtSz: 3017832
    lineRate: 6
    offRate: 4
    offMask: 0xfffffff0
    ftabChars: 10
    eftabLen: 0
    eftabSz: 0
    ftabLen: 1048577
    ftabSz: 4194308
    offsLen: 754458
    offsSz: 3017832
    lineSz: 64
    sideSz: 64
    sideGbwtSz: 48
    sideGbwtLen: 192
    numSides: 62872
    numLines: 62872
    gbwtTotLen: 4023808
    gbwtTotSz: 4023808
    reverse: 0
    linearFM: Yes
Total time for call to driver() for forward index: 00:00:04
total 69112
-rw-r--r--  2 bacon  staff  12273406 Mar 29 08:05 all-but-mito-genome.fa
-rw-r--r--  1 bacon  staff   8219399 Mar 29 08:11 all-but-mito-genome.fa.1.ht2
-rw-r--r--  1 bacon  staff   3017836 Mar 29 08:11 all-but-mito-genome.fa.2.ht2
-rw-r--r--  1 bacon  staff       152 Mar 29 08:11 all-but-mito-genome.fa.3.ht2
-rw-r--r--  1 bacon  staff   3017832 Mar 29 08:11 all-but-mito-genome.fa.4.ht2
-rw-r--r--  1 bacon  staff   5357645 Mar 29 08:11 all-but-mito-genome.fa.5.ht2
-rw-r--r--  1 bacon  staff   3071004 Mar 29 08:11 all-but-mito-genome.fa.6.ht2
-rw-r--r--  1 bacon  staff        12 Mar 29 08:11 all-but-mito-genome.fa.7.ht2
-rw-r--r--  1 bacon  staff         8 Mar 29 08:11 all-but-mito-genome.fa.8.ht2
-rw-r--r--  2 bacon  staff       375 Mar 29 08:05 all-but-mito-genome.fa.fai
```

```
# To save terminal output to a file:
shell-prompt: ./12-hisat2-index.sh |& tee hisat2-index.out

# If the syntax above doesn't work in your shell:
shell-prompt: ./12-hisat1-index.sh 2>&1 | tee hisat2-index.out
```

Note

Be sure to carefully review the output to the terminal for warnings, errors, and reasonable values. If you saved a copy of the terminal output by piping through **tee**, you can use tools such as **grep -i error file.out** to search for error messages, etc.

Building the HISAT2 index for a large genome could take hours. Once it completes, we can run the HISAT2 alignment stage for each of our trimmed read samples. This stage will require the most computing resources of any in this pipeline. As the transcriptome of most eukaryotic organisms is a small fraction of the size of the genome (only a few percent of our DNA contains coding sequences), mapping reads to a genome is far more expensive than mapping to the transcriptome.

HISAT2, like many bioinformatics tools, has sketchy support for modern compression tools at the time of this writing. It cannot input or output **xz** or **zstd** compressed files, though it does support **gzip**. Unlike most tools, it cannot read data from a pipe, either, not even a named pipe as we used with Kallisto. HISAT2 requires that the input is an ordinary file, as it performs *seek* operations to move about within the file, rather than just read through it sequentially. It is not possible to perform a seek operation on a pipe, since only a small portion of the file stream contents exist within it at any given moment. Unlike an ordinary file, when a process

reads from a pipe, the bytes it read are removed from the pipe, and it is no longer possible for a seek to "back up" to an earlier point within the stream. Hence, for this stage alone, we copy the **zstd** compressed FASTQ file to a **gzip** compressed file. You might, for simplicity, choose to use **gzip** throughout the pipeline, though doing so will slow down all other stages compared to using **zstd**.

Note We could also just create an uncompressed copy of the file, but this would use about twice as much disk space as the gzipped file. This increases the risk of a "disk full" error, and might slow down processing. Transferring twice as much data to and from disk is likely slower than running it through **gzip --fast** and **gunzip**, especially if the disk is on a remote file server.

Hisat2 outputs an unsorted SAM stream, which is bulky and not ideal for subsequent stages of our pipeline. We can use **samtools sort** to both sort the output and simultaneously convert it to compressed BAM format to save disk space. Since Hisat2 takes much longer than **samtools sort**, piping the Hisat2 output directly into samtools will sort the output without costing us any measurable extra time. (If you run **top** in another terminal while the alignment is running, you will see **samtools** using very little CPU time, which indicates that it is having no trouble keeping up with the data is is being fed by Hisat2.) If we write the Hisat2 output to a BAM file and sort it afterward, the whole process will take significantly longer, and will use far more disk space for the extra uncompressed alignment map. Below are the commands we can use to efficiently run a Hisat2 alignment.

```
# Create output directory for alignments
shell-prompt: mkdir Results/13-hisat2-align
shell-prompt: cd Results/13-hisat2-align

# Create a gzipped copy of the FASTQ input that HISAT2 can read.
# Use gzip --fast to minimize CPU time in exchange for a larger file.
shell-prompt: zstdcat ../05-trim/sample01-cond1-rep01-trimmed.fastq.zst \
    | gzip --fast > sample01-cond1-rep01-trimmed.fastq.gz

# Run hisat2 alignment, sending output to the standard output, and piping
# directly into "samtools sort" to sort the alignments for the
# quantification stage that comes next and compress to BAM format.
shell-prompt: hisat2 --threads 4 -x ../12-hisat2-index/all-but-mito-genome \
    -U sample01-cond1-rep01-trimmed.fastq.gz \
    | samtools sort > sample01-cond1-rep01-trimmed.bam

# samtools index is quick, so no need to work it into the pipe above,
# which would be a little complicated.
shell-prompt: samtools index sample01-cond1-rep01-trimmed.bam
```

The DIY-RNA-Seq-DE repository contains a script for aligning all of our trimmed read files. This script will align one sample at a time, using multiple processors for each job.

```
shell-prompt: ./13-hisat2-align

[ Output omitted for brevity ]

===
Converting ../../Results/05-trim/sample06-cond1-rep03-trimmed.fastq.zst to gzip format...
Running hisat2 sample06-cond1-rep03-trimmed.fastq.gz -> sample06-cond1-rep03-trimmed.bam...
+ hisat2 --threads 8 -x ../../Results/12-hisat2-index/all-but-mito-genome.fa -U sample06- ↩
    cond1-rep03-trimmed.fastq.gz
+ samtools sort
1225424 reads; of these:
  1225424 (100.00%) were unpaired; of these:
    30170 (2.46%) aligned 0 times
    1045312 (85.30%) aligned exactly 1 time
    149942 (12.24%) aligned >1 times
97.54% overall alignment rate
+ set +x
Indexing sample06-cond1-rep03-trimmed.bam...
```

```
# To save terminal output to a file:
shell-prompt: ./13-hisat2-align.sh |& tee hisat2-align.out

# If the syntax above doesn't work in your shell:
shell-prompt: ./13-hisat2-align.sh 2>&1 | tee hisat2-align.out
```

Note
Be sure to carefully review the output to the terminal for warnings, errors, and reasonable values. If you saved a copy of the terminal output by piping through **tee**, you can use tools such as **grep -i error file.out** to search for error messages, etc.

Scrolling back your terminal output to review, we should see a high alignment rate for every sample (assuming we are using a model organism with a fairly mature genome reference). The example above shows 85% of reads uniquely mapped and another 12% multi-mapped, which is pretty good. That leaves only 3% of the sample reads unaligned. Note that we omitted the mitochondrial sequences from our references, so if there is mitochondrial RNA in the samples, most of it will be unaligned. Omitting X and Y chromosomes from a mammalian RNA references will likewise reduce the alignment rate.

Verifying the correctness of millions of alignments is challenging. Some researchers have used simulated data to estimate the accuracy of the alignment software [Baruzzo], [Donato], though accuracy for one data set does not guarantee accuracy for all, due to differences in read length, redundancy in the genome, etc. We recommend trying out multiple aligners for each study and comparing the downstream results of the entire pipeline resulting from each.

As with every stage, we want to at least perform some basic sanity checks before moving on. A simple listing of the output directory is a good place to start. We see below that all of the output files are of similar size (and none are 0), so there is no reason to suspect that any of the jobs failed.

```
shell-prompt: l Results/13-hisat2-align/*.bam
-rw-r--r--  1 bacon bacon   54M Apr 11 08:08 Results/13-hisat2-align/sample01-cond1-rep01- ↩
    trimmed.bam
-rw-r--r--  1 bacon bacon   89M Apr 11 08:09 Results/13-hisat2-align/sample02-cond2-rep01- ↩
    trimmed.bam
-rw-r--r--  1 bacon bacon   75M Apr 11 08:09 Results/13-hisat2-align/sample03-cond2-rep02- ↩
    trimmed.bam
-rw-r--r--  1 bacon bacon   76M Apr 11 08:10 Results/13-hisat2-align/sample04-cond2-rep03- ↩
    trimmed.bam
-rw-r--r--  1 bacon bacon   72M Apr 11 08:10 Results/13-hisat2-align/sample05-cond1-rep02- ↩
    trimmed.bam
-rw-r--r--  1 bacon bacon   61M Apr 11 08:10 Results/13-hisat2-align/sample06-cond1-rep03- ↩
    trimmed.bam
```

Following the "know your data" principle, we can have a look at the SAM output. The official specification for the SAM file format can be found at https://samtools.github.io/hts-specs/. A more concise description, suitable for quickly reviewing this output, is available at https://en.wikipedia.org/wiki/SAM_(file_format).

```
shell-prompt: samtools view Results/13-hisat2-align/sample01-cond1-rep01-trimmed.bam|more
ERR458493.889005        256     1       1854    1       2S47M   *       0       0       ↩
    GTTGGCTGGCTTTAATCTGCTGGAGTACCATGGAACACCGGTGATCATT       =4=22AC; ↩
    CFFIIIEGICHGIIIBIGHHHFAECGGHIBFA@?@BFC>FB       AS:i:-2 ZS:i:-2 XN:i:0  XM:i:0  XO:i:0   ↩
    XG:i:0  NM:i:0  MD:Z:47 YT:Z:UU NH:i:5
ERR458493.579447        272     1       1930    1       47M     *       0       0       ↩
    CAACATGGTGGTGAAGTCACCGTAGTTGAAAACGGCTTCAGCAACTT CFIFFEIF@@IIIFFFIIGIIIIIIHFHFGF? ↩
    IIFIIIFFFFFDDB= AS:i:0  ZS:i:0  XN:i:0  XM:i:0  XO:i:0  XG:i:0  NM:i:0  MD:Z:47 YT:Z:UU  ↩
    NH:i:5
ERR458493.918931        272     1       1930    1       48M     *       0       0       ↩
    CAACATGGTGGTGAAGTCACCGTAGTTGAAAACGGCTTCAGCAACTTC        DCF@>HF? ↩
    EIIJHIIGEFBBIIJJJIHJIGEHJIJIJIFDHFFDDDA=        AS:i:0  ZS:i:0  XN:i:0  XM:i:0  XO:i:0   ↩
    XG:i:0  NM:i:0  MD:Z:48 YT:Z:UU NH:i:5
ERR458493.784623        16      1       1934    1       51M     *       0       0       ↩
    ATGGTGGTGAAGTCACCGTAGTTGAAAACGGCTTCAGCAACTTCGACTGGG      ↩
    IIIIIIIIGIGIIHFIEGF@IGIIIIIHIIIIIIIIIIHHHHHDDDD?=;@     AS:i:0  ZS:i:0  XN:i:0  XM:i:0   ↩
    XO:i:0  XG:i:0  NM:i:0  MD:Z:51 YT:Z:UU NH:i:5
```

```
[ Output omitted for brevity ]
```

Note that the SAM format is not really meant to be human-readable, so it will take some work to decipher the output. It is not critical that you understand SAM files at this point, but it may come in handy down the road if you become a bioinformatician.

The **samtools quickcheck** command can flag any obvious problems with the SAM/BAM files, such as invalid FLAGS, problems with file formatting, etc. This command can take multiple filenames as arguments. There will be no output if no problems are found in the file, in line with "no news is good news" convention for Unix commands.

```
shell-prompt: samtools quickcheck Results/13-hisat2-align/*.bam
```

The number of alignments for each sample will vary, so a simple line count like the one we used for the Kallisto abundance files will not be useful with out SAM outputs.

4.11.3 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

---

1. What is the difference between "read mapping" and "alignment"?
2. What is an alignment map?
3. What is the purpose of an index in read mapping?
4. Show the commands for generating a kallisto index called `kallisto-mouse.index` from a reference called `mouse-transc fasta`.
5. How do we determine the success of **kallisto index** to an extent that justifies moving on to the next step of performing pseudo-alignments and quantification?
6. What is multithreading?
7. What is a named pipe, and when are they needed?
8. What, in general, are the advantages of a named pipe (or an unnamed pipe for that matter) over using a regular file for intermediate results?
9. What are three simple commands to perform sanity checks for Kallisto outputs in the directory ./Kallisto-out?
10. What is one reason to map reads to a genome as well as to a transcriptome?
11. What is the main problem when mapping mRNA reads to a eukaryotic genome? What is the solution?
12. Show the commands needed to build a Hisat2 index in ./Results/Hisat2 from the reference genome ./Results/Reference/genome.fa.
13. Show the commands needed to map reads from ./Results/Trimmed/sample01-cond1-rep01.fastq.gz to the reference described in the previous question, storing the alignment map in ./Results/Hisat2/sample01-cond1-rep01.bam.
14. What are two simple sanity checks we can perform on our Hisat2 alignment output?

4.12 Differential expression analysis software

As stated in Section 2.9, the differential analysis stage of an RNA-Seq pipeline tends to be the most challenging in terms of software tools. While the prior stages are mostly just a matter of installing software and running it with existing files as inputs, differential analysis with the most popular traditional tools requires significant knowledge of R programming, including data structures such as matrices and data frames.

For our original RNA-Seq analysis in 2019, we employed Sleuth [Sleuth], developed by the same lab as Kallisto. Like most differential analysis tools, Sleuth requires the user to write a fairly sophisticated R language script in order to use it. Our Sleuth script contains over 500 lines of total text. According to the **cloc** (Count Lines Of Code) command, this includes 279 lines of actual Rscript code and 128 lines of comments, the rest being blank lines.

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
R                                1            113            128            279
-------------------------------------------------------------------------------
```

There is some redundant code that could be factored out to significantly reduce size of the script, but this would require even more sophisticated R programming skills in order to generalize the code to work with any sample.

Another issue is the fact that, as of the time of this writing (spring 2025), there has been almost no development activity in the Sleuth project since 2018. This is unfortunately common among scientific software projects. The developers are often scientists in the "publish or perish" world, developing the software for the purpose of publishing a paper. Once the paper is done, they need to move on to other projects and have little or no time to maintain the software. In the case of Sleuth, the lack of maintenance led to installation issues for many users, with the published installation instructions non-functional at times.

One alternative would be to use DESeq2, a very popular and more actively maintained differential analysis tool. Unfortunately, DESeq2 also requires the user to know R programming for basic use. The DESeq2 manual is 80+ pages, the "Quick Start" section of which assumes that the reader understands R data structures such as matrices and tables. We have also experienced difficulties installing DESeq2 on Redhat Enterprise Linux derivatives, which are used on most HPC clusters, using the recommended Bioconductor package manager. Errors of the type we encountered are described in Chapter 3. These are highly technical I.T. issues that most biologists would not be able to solve on their own.

There is also a recent reimplementation of DESeq2 in Python rather than R (https://pypi.org/project/pydeseq2/). The Python implementation will likely have fewer installation hurdles (see the comments on R package installation issues in Chapter 3), but it is no easier to *use* than the original R language version. It requires the same kind of programmatic approach, just in the Python language and data structures instead of R.

In addition to installation and ease of use issues, existing differential analysis tools have been shown to suffer from high *false discovery rate* (*FDR*) and poor performance for large sample counts like those often found in population studies [Li-FalsePos].

To solve all of these issues for this stage of the analyses, we created an entirely new tool called *FASDA* (Fast and Simple Differential Analysis). Compared to the most popular tools, FASDA has the following advantages:

- FASDA can be installed and updated quickly, easily, and reliably on any Unix-like system using FreeBSD ports, dreckly, and any other package manager for which someone maintains a package.
- Using FASDA does not require any programming skills. Rather, FASDA provides simple Unix commands that work directly with the abundance outputs of the Kallisto aligner/quantifier, or with the SAM/BAM alignment map of traditional aligners.
- FASDA controls the false discovery rate (tested by computing P-values for large randomized samples) by computing exact P-values for small sample sizes and using the non-parametric Mann-Whitney U-test (also known as Wilcoxon rank sum test) for eight or more replicates [Li-FalsePos]. (A non-parametric test does not make assumptions about the data distribution, leading to better stability.)
- FASDA is written entirely in C, which is roughly 100 times as fast as interpreted languages like R or Python, and the code is highly optimized to ensure the best possible performance. This is irrelevant for typical three-sample RNA analyses, which run quickly with any DE tool, but could be important for studies with many samples.

FASDA currently uses the same normalization method (median ratio normalization) as DESeq2, and the normalized counts and fold-changes have been verified to agree closely with those from DESeq2. Alternative normalization and P-value estimation methods may be added in the future. The details of using FASDA are described in the following sections.

Well-established differential analysis tools commonly report very different sets of differentially expressed features. [Li-FalsePos] reported that 23% to 75% of the SDE (significantly differentially expressed) features identified by DESeq2 were missed by edgeR. In one data set tested, DESeq2 and edgeR had only an 8% overlap in the SDE features they identified.

Popular differential analysis tools such as DESeq2 and edgeR assume a negative binomial distribution [Schurch], [Li-FalsePos]. While this assumption has some advantages like improving statistical power, it does not cope well with skewed data and outliers [Li-FalsePos]. To avoid these issues, FASDA computes exact P-values for data with less than eight replicates and uses the non-parametric Mann-Whitney (Wilcoxon rank sum) test to estimate P-values for eight or more replicates. As non-parametric methods, neither approach makes any assumptions about the data distribution. These methods are more robust in handling skewed data that do not fit an assumed distribution well, like those typical in RNA-Seq [Li-FalsePos].

4.12.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

---

1. What is the biggest problem for the average biologist running differential expression (DE) software?
2. What normalization method does FASDA use?
3. How consistent are the results across popular DE tools?
4. What is the advantage of using a non-parametric statistical method?

4.13 Quantification

Read mapping, as performed by Hisat2, only determines the location in the genome from which a read likely originated. In order to perform a differential analysis, we still need to estimate how many reads came from each feature (e.g. gene or transcript). This process is known as *quantification*, or *abundance estimation*. Note that what we *really* would like to know is the number of *transcripts* from each condition, not the number of *reads*. A read is generally a small part of a transcript, and other parts of the transcript might not even have been sequenced. However, since the ratio of reads / transcripts is due largely to random technical factors unrelated to the biological conditions we are comparing, we assume the ratio is roughly the same for both conditions, especially when the read counts for the feature are high (at least dozens, sometimes hundreds or thousands). I.e., reads1 / transcripts1 will be about the same as reads2 / transcripts2, where 1 and 2 represent the two biological conditions being compared. Simple simple algebra shows that this also implies reads1 / reads2 will be about the same as transcripts1 / transcripts2, so determining transcript counts will not change anything for our purpose here.

Kallisto performs its own quantification, and in fact does not even output a SAM/BAM alignment map. Older versions could optionally produce an alignment map, but this feature has been deprecated. The primary output from Kallisto is a tab-separated file called `abundances.tsv`, which contains the estimated counts for each feature in the transcriptome along with some other statistics.

Other aligners, such as Bowtie2, BWA, HISAT2, and STAR, only produce an alignment map in SAM or BAM format, indicating a best guess as to the origin of each read within the genome or transcriptome. The certainty of the location of each read is encoded in the alignment map as an *alignment score*, which can be used by the abundance estimator to decide whether to count the alignment. In many cases, the aligner will find only one good match in the transcriptome or genome. In some cases, it may find two or more locations that are about equally likely, in which case we may not be able to count the read with much confidence.

The **fasda abundance** command uses a SAM or BAM alignment map and a *GFF3* (*General Feature Format version 3*) file, which lists locations of known genes, transcripts, exons, etc. Each alignment location in the alignment mapped it looked up in the feature file, and the count is incremented for the matching feature. The **fasda abundance** command produces an abundance file in the same format as Kallisto's. By standardizing the format of the abundance file this way, we ensure that the next analysis

stage can use the exact same Unix commands, regardless of which aligner was used. This eliminates the need for researchers to write code that manipulates the output of the aligners into a form suitable for the differential analysis tool.

At the time of this writing, **fasda abundance** employs stringtie to estimate abundances, and then reformats the output into a kallisto-like abundance file, so that the next stage of the pipeline is independent of the aligner used. Future versions of **fasda abundance** may support using other abundance tools as well.

Below is a sample **fasda abundance** command from the FASDA Yeast-test data. It produces an abundance file called `Results/13-hisat2-align/sample01-cond1-rep01-trimmed-abundance.tsv`, (based on the name of the BAM file). The first argument is the read length (which cannot be determined automatically from an abundance file), the second is the GFF3 feature file describing known genes, transcripts, etc., and the third is the SAM or BAM output file from hisat2-align.

```
# Estimate abundances for the sample 1 alignments
shell-prompt: fasda abundance 51 \
    Results/08-reference/Saccharomyces_cerevisiae.R64-1-1.106.gff3 \
    Results/13-hisat2-align/sample01-cond1-rep01-trimmed.bam

# View the results
shell-prompt: head Results/13-hisat2-align/sample01-cond1-rep01-trimmed-abundance.tsv
target_id       length  eff_length      est_counts      tpm     fpkm
ETS1-1_rRNA     700     0.0     57.6    74.374039       83.065010
ETS1-2_rRNA     700     0.0     57.6    74.374039       83.065010
ETS2-1_rRNA     211     0.0     0.0     0.000000        0.000000
ETS2-2_rRNA     211     0.0     0.0     0.000000        0.000000
HRA1_ncRNA      564     0.0     2.1     3.356013        3.748179
ICR1_ncRNA      3199    0.0     21.0    5.927928        6.620635
IRT1_ncRNA      1489    0.0     16.8    10.183536       11.373530
ITS1-1_rRNA     361     0.0     9.9     24.798952       27.696829
ITS1-2_rRNA     361     0.0     10.5    26.203169       29.265133
```

Note how the format of the output is the same as a Kallisto abundance file. At the time of this writing, FASDA does not compute effective lengths, as they are not needed for subsequent stages of a differential analysis. This may change in future FASDA releases.

The DIY-RNA-Seq-DE repository includes a script for computing abundances for all HISAT2 outputs:

```
shell-prompt: ./14a-fasda-abundance-hisat2.sh

[ Output omitted for brevity ]

+ fasda abundance --output-dir . 51 ../../Results/08-reference/Saccharomyces_cerevisiae.R64 ↩
    -1-1.106.gff3 ../../Results/13-hisat2-align/sample06-cond1-rep03-trimmed.bam

Writing abundances to ./sample06-cond1-rep03-trimmed-abundance.tsv
Writing GTF to ./sample06-cond1-rep03-trimmed.gtf
Running stringtie -e -G ../../Results/08-reference/Saccharomyces_cerevisiae.R64-1-1.106. ↩
    gff3 -o ./sample06-cond1-rep03-trimmed.gtf -A ./sample06-cond1-rep03-trimmed-stringtie- ↩
    genes.tsv ../../Results/13-hisat2-align/sample06-cond1-rep03-trimmed.bam...
+ set +x
target_id       length  eff_length  est_counts  tpm           fpkm
ETS1-1_rRNA     700     0.0         89.6        102.589455    117.042557
ETS1-2_rRNA     700     0.0         89.6        102.592270    117.045776
ETS2-1_rRNA     211     0.0         0.3         1.303424      1.487054
ETS2-2_rRNA     211     0.0         0.3         1.303424      1.487054
HRA1_ncRNA      564     0.0         0.1         0.086043      0.098165
ICR1_ncRNA      3199    0.0         27.5        6.886437      7.856618
IRT1_ncRNA      1489    0.0         32.7        17.583792     20.061049
ITS1-1_rRNA     361     0.0         22.1        49.068081     55.980938
ITS1-2_rRNA     361     0.0         22.0        48.822964     55.701290
    6982   41892  323625 sample06-cond1-rep03-trimmed-abundance.tsv
```

```
# To save terminal output to a file:
```

```
shell-prompt: ./14a-fasda-abundance-hisat2.sh |& tee fasda-abundance-hisat2.out

# If the syntax above doesn't work in your shell:
shell-prompt: ./14a-fasda-abundance-hisat2.sh 2>&1 | tee fasda-abundance-hisat2.out
```

Note
Be sure to carefully review the output to the terminal for warnings, errors, and reasonable values. If you saved a copy of the terminal output by piping through **tee**, you can use tools such as **grep -i error file.out** to search for error messages, etc.

The quality checks to perform here are the same as those described at the end of Section 4.11.1.

4.13.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

---

1. What is quantification?
2. Do we need to convert our read counts to transcript counts for the purpose of differential expression analysis?
3. What are the inputs to a quantification program?
4. Show a FASDA command to compute abundances for each transcript in the alignment map `./Results/Align/sample01-cond1-rep1.bam` using the feature file `./Results/Reference/t-rex.gff3`. The raw reads are 100 bases long.

4.14 Normalizing across samples

The amount of DNA or RNA in each biological sample (library) can vary immensely due to countless uncontrollable variables in the lab. This directly impacts the read counts produced when sequencing. If there are twice as many total RNA molecules in sample 1 as there are in sample 2, then we would expect about twice as many reads to map to a feature from sample 1, assuming the same level of actual gene expression occurred in both samples. I.e., a 2 to 1 ratio of abundances here would indicate *the same level* of gene expression. What we want is a 1:1 ratio of reads where there was no change in expression between samples. Hence, the read counts must be *normalized* to compensate for different library sizes before they can be directly compared across samples [Evans]. Some normalization methods rely on biological techniques such as *spike-in controls*, as mentioned in Section 2.8.

Data show that there can be significant variation of read counts even when sequencing equal partitions of the same sample solution in the same sequencing run, known as *technical replicates*. Sequencing machines allow processing multiple samples simultaneously, which eliminates some potential variation in comparing technical replicates. I.e., there are fewer variables among samples sequenced together than among those sequences in separate runs. Figure 4.32 shows variation in Kallisto read counts from seven technical replicates of yeast data produced in the same sequencing run (each replicate was sequences in a different *lane* in an Illumina sequencer). Note how the counts for YLL050C_mRNA range from 194 to 282, even though they come from the same biological sample and the same sequencing run.

```
===> Results/Tech-reps/kallisto-quant/ERR458493/abundance.tsv
target_id       length  eff_length  est_counts   tpm
YPL071C_mRNA    471     282         14           35.3946
YLL050C_mRNA    432     243         240          704.147
YMR172W_mRNA    2160    1971        43.6935      15.8048
YOR185C_mRNA    663     474         44           66.1809

===> Results/Tech-reps/kallisto-quant/ERR458494/abundance.tsv
target_id       length  eff_length  est_counts   tpm
YPL071C_mRNA    471     282         17           43.3106
YLL050C_mRNA    432     243         261          771.664
YMR172W_mRNA    2160    1971        39.4246      14.3706
YOR185C_mRNA    663     474         41           62.1441

===> Results/Tech-reps/kallisto-quant/ERR458495/abundance.tsv
target_id       length  eff_length  est_counts   tpm
YPL071C_mRNA    471     282         13           33.329
YLL050C_mRNA    432     243         249          740.835
YMR172W_mRNA    2160    1971        54           19.8078
YOR185C_mRNA    663     474         43           65.5871

===> Results/Tech-reps/kallisto-quant/ERR458496/abundance.tsv
target_id       length  eff_length  est_counts   tpm
YPL071C_mRNA    471     282         13           35.9038
YLL050C_mRNA    432     243         245          785.246
YMR172W_mRNA    2160    1971        39.1726      15.479
YOR185C_mRNA    663     474         37           60.7952

===> Results/Tech-reps/kallisto-quant/ERR458497/abundance.tsv
target_id       length  eff_length  est_counts   tpm
YPL071C_mRNA    471     282         13           42.0872
YLL050C_mRNA    432     243         194          728.872
YMR172W_mRNA    2160    1971        31.4292      14.558
YOR185C_mRNA    663     474         29           55.8567

===> Results/Tech-reps/kallisto-quant/ERR458498/abundance.tsv
target_id       length  eff_length  est_counts   tpm
YPL071C_mRNA    471     282         17           55.287
YLL050C_mRNA    432     243         206          777.471
YMR172W_mRNA    2160    1971        22.5807      10.5069
YOR185C_mRNA    663     474         37           71.589

===> Results/Tech-reps/kallisto-quant/ERR458499/abundance.tsv
target_id       length  eff_length  est_counts   tpm
YPL071C_mRNA    471     282         14           34.8473
YLL050C_mRNA    432     243         282          814.579
YMR172W_mRNA    2160    1971        44.8517      15.9728
YOR185C_mRNA    663     474         53           78.4852
```

Figure 4.32: Variance among technical replicates

Note that the estimated counts for YLL050C_mRNA range from 194 to 282, when in theory they should be the same. Technical variation of this magnitude does not necessarily indicate poor lab technique in dividing the sample, though that would be one possible explanation. There are many other factors beyond the control of the lab biologist that contribute to technical variation.

Normalization methods often mentioned in publications, such as *Transcripts Per Million* (*TPM*) and *Fragments Per Kilobase Million* (*FPKM*) are not useful for differential analysis. This is because they perform *within sample normalization*, which normalizes counts for features of different lengths within the same sample. The goal of within sample normalization is to allow comparison of counts across genes of different lengths. Since reads from the sequencer are all the same length, a longer gene

will have more reads mapped to it than a shorter one that produced the same number of transcripts.

For differential expression analysis, we perform *between sample normalization*, which normalizes counts for the *same feature* across different samples. Since we are always comparing counts from the same feature, we do not need to adjust for different feature lengths. The ratio of reads / transcripts is the same for both samples because the length of the feature is the same for both samples. Hence, it doesn't matter whether we count transcripts or reads, and we can just save ourselves the effort of adjusting for feature length.

Of the many normalization methods available, we chose *median ratio normalization*, (*MRN*) due to its simplicity and robustness [Evans]. It is the same method used by DESeq2, one of the most popular DE tools. It is not critical for you to understand median ratio normalization at this point, but since it is relatively simple, the procedure is described below for anyone who is interested.

1. Compute the average of log(count) for all replicates for each feature. Features with an average of -infinity are discarded. This occurs where the count was zero for any replicate, since the average of any set of values containing -infinity is -infinity. Features with very low read counts (or an outlier of zero for any sample) will not likely produce reliable fold-changes.

```
# Raw counts
Feature          R1  R2  R3
YPL071C_mRNA    102  84  97
YOR185C_mRNA     31  55  28

# Log counts
Feature           R1   R2   R3   Average
YPL071C_mRNA    4.62 4.43 4.57  4.54
YOR185C_mRNA    3.43 4.01 3.33  3.59
```

2. Subtract the average log(counts) from every log(count) for that feature.

```
# log(count) - average
Feature          R1     R2    R3
YPL071C_mRNA    0.08 -0.11  0.03
YOR185C_mRNA   -0.16  0.42 -0.26
```

   Note that subtracting the logs of two values effectively computes the ratio of the values. This is where the name "Median Ratio Normalization" comes from. Recall from basic algebra:

```
log(A) - log(B) = log(A / B)

or

x ^ A / x ^ B = x ^ (A - B)
```

3. Keep the median of the log(ratios) *for each sample* and discard the rest. We call this median MR_{sample}. MR_{sample} is higher for samples with overall higher counts, which we presume are proportional to the DNA library size (the number of DNA/RNA fragments in the sample).

```
# Median ratio for each replicate
                R1      R2     R3
Median       -0.04   0.155 -0.115
```

4. Reverse the earlier transformation into log space, by computing $\exp(MR_{sample})$. Then divide all counts from that sample by this factor. This scales counts from each sample differently, effectively equalizing the library sizes of all samples, so that counts from different samples can be compared to each other with less influence from differences in library size. These are roughly the counts we would expect if all samples had the same library size (total RNA/DNA fragment counts).

```
# Scaling factors
0.96  1.16  0.89    exp(-0.04)  exp(0.155)  exp(-0.115)

# Raw counts
102  84  97
 31  55  28
```

```
# Normalized counts
106.25  87.5  108.99    102/0.96  84/1.16  97/0.89
32.29   47.41 31.46     31/0.96  55/1.16  28/0.89
```

Every statistical normalization method depends on some assumptions [Evans]. For our example, we normalize counts across *all* samples, including all conditions, to compensate for the different library sizes of each sample. This approach only works well if the counts are fairly *symmetric* across conditions, meaning that we have about the same number of up-regulated and down-regulated genes [Evans]. This is assumed to be the case for most transcript libraries. If we encounter the rare situation where most genes are up-regulated in one condition (known as a *global shift*), then normalizing across conditions will effectively erase much of the real fold-change.

The inputs to **fasda normalize** are separate files containing estimated counts for each sample, following the format of Kallisto abundance files. If you are not using Kallisto, these input files can be generated from SAM/BAM alignment maps using **fasda abundance**, as described in Section 4.13. The output is a single tab-separated file with normalized counts for all replicates of one feature on each line.

```
shell-prompt: mkdir Results/11-fasda-kallisto
shell-prompt: fasda normalize \
    --output Results/11-fasda-kallisto/all-norm.tsv \
    Results/10-kallisto-quant/sample01-cond1-rep01-trimmed/abundance.tsv \
    Results/10-kallisto-quant/sample02-cond1-rep02-trimmed/abundance.tsv \
    Results/10-kallisto-quant/sample03-cond1-rep03-trimmed/abundance.tsv \
    Results/10-kallisto-quant/sample04-cond2-rep01-trimmed/abundance.tsv \
    Results/10-kallisto-quant/sample05-cond2-rep02-trimmed/abundance.tsv \
    Results/10-kallisto-quant/sample06-cond2-rep03-trimmed/abundance.tsv

shell-prompt: mkdir Results/14-fasda-hisat2
shell-prompt: fasda normalize \
    --output Results/14-fasda-hisat2/all-norm.tsv \
    Results/14-hisat2-quant/sample01-cond1-rep01-trimmed-abundance.tsv \
    Results/14-hisat2-quant/sample02-cond1-rep02-trimmed-abundance.tsv \
    Results/14-hisat2-quant/sample03-cond1-rep03-trimmed-abundance.tsv \
    Results/14-hisat2-quant/sample04-cond2-rep01-trimmed-abundance.tsv \
    Results/14-hisat2-quant/sample05-cond2-rep02-trimmed-abundance.tsv \
    Results/14-hisat2-quant/sample06-cond2-rep03-trimmed-abundance.tsv
```

The DIY-RNA-Seq-DE repository provides scripts for normalizing counts across all samples.

```
shell-prompt: ./11a-fasda-normalize-kallisto.sh

[ Output omitted for brevity ]

All samples normalized counts:

YPL071C_mRNA    20.622863       32.000000       36.246158       33.516861       29.586034  ↩
          30.160359
YLL050C_mRNA    353.534800      434.000000      416.830814      551.385227      559.669150  ↩
         531.157429
YMR172W_mRNA    64.363221       62.437500       56.559593       80.171016       83.758885  ↩
          86.110335
YOR185C_mRNA    64.814713       34.000000       73.700521       48.632308       59.172069  ↩
          61.158505
YLL032C_mRNA    20.622863       40.000000       41.078979       22.344574       16.436686  ↩
          25.971420
```

```
# To save terminal output to a file:
shell-prompt: ./11a-fasda-normalize-kallisto.sh |& tee fasda-normalize-kallisto.out

# If the syntax above doesn't work in your shell:
shell-prompt: ./11a-fasda-normalize-kallisto.sh 2>&1 | tee fasda-normalize-kallisto.out
```

```
shell-prompt: ./14b-fasda-normalize-hisat2.sh

[ Output omitted for brevity ]

All samples normalized counts:

ETS1-1_rRNA     84.155084       91.000000       106.314886      102.613421      102.624206  ↩
          86.962822
ETS1-2_rRNA     84.155084       91.000000       106.314886      102.613421      102.624206  ↩
          86.962822
ETS2-1_rRNA     0.000000        1.000000        0.357963        1.055829        0.000000  ↩
    0.916275
ETS2-2_rRNA     0.000000        1.000000        0.357963        1.055829        0.000000  ↩
    0.916275
HRA1_ncRNA      3.078844        0.100000        0.119321        0.659893        0.000000  ↩
    0.832977
```

```
# To save terminal output to a file:
shell-prompt: ./14b-fasda-normalize-hisat2.sh |& tee fasda-normalize-hisat2.out

# If the syntax above doesn't work in your shell:
shell-prompt: ./14b-fasda-normalize-hisat2.sh 2>&1 | tee fasda-normalize-hisat2.out
```

Note

Be sure to carefully review the output to the terminal for warnings, errors, and reasonable values. If you saved a copy of the terminal output by piping through **tee**, you can use tools such as **grep -i error file.out** to search for error messages, etc.

The normalized counts file should have exactly the same number of features as the sample abundance files from which it was derived. Note that the abundance files have a header line describing the columns, while the normalized counts file does not, so the Unix **wc** command will show one line fewer for the abundance file.

```
shell-prompt: wc -l Results/14-fasda-hisat2/all-norm-3.tsv Results/14-fasda-hisat2/*- ↩
    abundance.tsv
     6981   Results/14-fasda-hisat2/all-norm-3.tsv
     6982   Results/14-fasda-hisat2/sample01-cond1-rep01-trimmed-abundance.tsv
     6982   Results/14-fasda-hisat2/sample02-cond1-rep02-trimmed-abundance.tsv
     6982   Results/14-fasda-hisat2/sample03-cond1-rep03-trimmed-abundance.tsv
     6982   Results/14-fasda-hisat2/sample04-cond2-rep01-trimmed-abundance.tsv
     6982   Results/14-fasda-hisat2/sample05-cond2-rep02-trimmed-abundance.tsv
     6982   Results/14-fasda-hisat2/sample06-cond2-rep03-trimmed-abundance.tsv
    48873  total
```

Another quick sanity check should show that the abundance files from different aligners or quantifiers have about the same number of features. They may differ slightly due to some counts being zero (leading to elimination of the feature) from one aligner and not the other, but they should be very close.

```
shell-prompt: wc -l Results/11-fasda-kallisto/all-norm.tsv Results/14-fasda-hisat2/all-norm ↩
    .tsv
     6584 Results/11-fasda-kallisto/all-norm.tsv
     6981 Results/14-fasda-hisat2/all-norm.tsv
    13565 total
```

Normalized counts for the same feature from Kallisto may differ from those from Hisat2, but the ratios produced by a given quantifier should be about the same. For example, in the data below, 32.0 / 20.6 = 1.55 and 40.5 / 25.8 = 1.57. While we can spot check that here, it is easier to compare the fold-changes en masse after running **fasda fold-change** in the next stage of our pipeline.

```
shell-prompt: grep YPL071C_mRNA Results/11-fasda-kallisto/all-norm.tsv Results/14-fasda- ↩
    hisat2/all-norm.tsv
Results/11-fasda-kallisto/all-norm.tsv:YPL071C_mRNA     20.622863       32.000000       ↩
    36.246158       33.516861       29.586034       30.160359
Results/14-fasda-hisat2/all-norm.tsv:YPL071C_mRNA       25.803649       40.500000       ↩
    60.614997       46.126548       40.572361       45.813746
```

4.14.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. Why must we normalize counts across samples?
2. Does high technical variation indicate bad lab technique?
3. Why do we not use standard normalization methods such as TPM for differential analysis?
4. Show a FASDA command to normalize counts in the files `Results/Abundance/sample01-cond1-rep01-abundance.tsv`, `Results/Abundance/sample02-cond1-rep02-abundance.tsv`, `Results/Abundance/sample03-co.tsv`, and `Results/Abundance/sample04-cond2-rep02-abundance.tsv`, placing the normalized counts in `./normalized.tsv`.
5. What is a quick sanity check to verify that the normalization command did not fail?

4.15 Computing fold-changes and P-values

4.15.1 Running fasda fold-change

With normalized counts in-hand, we can finally proceed to the differential expression stage using **fasda fold-change**. This command takes as input two or more files produced by **fasda normalize**, each containing normalized counts for all samples from the same condition. The requirement of separate files for each condition, even though **fasda normalize** outputs a single file with all conditions, was a deliberate design decision to help users avoid mistakes. Separating the conditions into individual files is a simple and deliberate act that requires the user to focus on this important task. This makes it harder to accidentally use the wrong counts for a given condition. Extracting the appropriate columns from the single normalized counts file is extremely easy using the standard Unix **cut** command.

```
shell-prompt: mkdir Results/11-fasda-kallisto
shell-prompt: cd Results/11-fasda-kallisto
# Separate normalized counts by condition to avoid confusion
# Copy the feature name and columns 2, 3, and 4 to the condition 1 counts
# Copy the feature name and columns 5, 6, and 7 to the condition 2 counts
shell-prompt: cut -f 1-4 all-norm.tsv > cond1-norm-3.tsv
shell-prompt: cut -f 1,5-7 all-norm.tsv > cond2-norm-3.tsv
shell-prompt: fasda fold-change --output fc-3.txt \
                    cond1-norm-3.tsv cond2-norm-3.tsv
```

Caution

It is imperative that you separate the correct columns and place all counts from the same condition into each file. The columns in the normalized counts file are determined by the order of the filenames in the **fasda normalize** command. Column 1 is always the feature name. In the example below, the first three files are from condition 1, so columns 2, 3, and 4 in the normalized counts file are from condition 1, and columns 5, 6, and 7 are condition 2.

```
shell-prompt: fasda normalize \
    --output Results/11-fasda-kallisto/all-norm.tsv \
    Results/10-kallisto-quant/sample01-cond1-rep01-trimmed/abundance.tsv \
    Results/10-kallisto-quant/sample02-cond1-rep02-trimmed/abundance.tsv \
    Results/10-kallisto-quant/sample03-cond1-rep03-trimmed/abundance.tsv \
    Results/10-kallisto-quant/sample04-cond2-rep01-trimmed/abundance.tsv \
    Results/10-kallisto-quant/sample05-cond2-rep02-trimmed/abundance.tsv \
    Results/10-kallisto-quant/sample06-cond2-rep03-trimmed/abundance.tsv
```

The DIY-RNA-Seq-DE repository provides scripts for running **fasda fold-change** on all Kallisto and HISAT2 normalized counts. These scripts are carefully matched to the **fasda normalize** scripts to ensure that the counts are properly separated by condition.

```
shell-prompt: ./11b-fasda-fc-kallisto.sh

Condition 1 normalized counts:
YPL071C_mRNA    20.594876       32.000000       36.236210
YLL050C_mRNA    350.112900      433.000000      415.508536
YMR172W_mRNA    62.731406       62.437500       56.544069
YOR185C_mRNA    64.726755       33.000000       73.680293
YLL032C_mRNA    20.594876       40.000000       41.067704

Condition 2 normalized counts:
YPL071C_mRNA    33.528561       29.624418       30.173662
YLL050C_mRNA    546.318312      556.280730      528.877250
YMR172W_mRNA    80.199001       83.022430       86.639484
YOR185C_mRNA    48.649284       59.248835       60.347325
YLL032C_mRNA    22.352374       16.458010       25.982876

Computing fold-change for 3 replicates...
time fasda fold-change --output fc-3.txt cond1-norm-3.tsv cond2-norm-3.tsv

Feature                  MNC1    MNC2  SD/C1  SD/C2  FC 1-2 log2(FC) P-val
YPL071C_mRNA             29.6    31.1    0.3    0.2    1.05    0.07  0.72611
YLL050C_mRNA            399.5   543.8    0.2    0.2    1.36    0.44  0.02857
YMR172W_mRNA             60.6    83.3    0.2    0.2    1.38    0.46  0.02167
YOR185C_mRNA             57.1    56.1    0.3    0.2    0.98   -0.03  0.92562
YLL032C_mRNA             33.9    21.6    0.3    0.3    0.64   -0.65  0.14138

[ Output omitted for brevity ]

File                        Features   P < 0.05
fc-3.txt:                       6585       1027
```

```
# To save terminal output to a file:
shell-prompt: ./11b-fasda-fc-kallisto.sh |& tee fasda-fc-kallisto.out

# If the syntax above doesn't work in your shell:
shell-prompt: ./11b-fasda-fc-kallisto.sh 2>&1 | tee fasda-fc-kallisto.out
```

```
shell-prompt: ./14c-fasda-fc-hisat2.sh

Condition 1 normalized counts:
ETS1-1_rRNA     82.367568       89.699997       104.332540
```

```
ETS1-2_rRNA     82.367568       89.699997       104.332540
ETS2-1_rRNA     0.000000        1.000000        0.358121
ETS2-2_rRNA     0.000000        1.000000        0.358121
HRA1_ncRNA      3.077792        0.100000        0.119374

Condition 2 normalized counts:
ETS1-1_rRNA     101.198593      101.488767      86.000475
ETS1-2_rRNA     101.198593      101.488767      86.000475
ETS2-1_rRNA     1.055526        0.000000        0.833338
ETS2-2_rRNA     1.055526        0.000000        0.833338
HRA1_ncRNA      0.659704        0.000000        0.833338

Computing fold-change for 3 replicates...
time fasda fold-change --output fc-3.txt cond1-norm-3.tsv cond2-norm-3.tsv

Feature                 MNC1    MNC2  SD/C1  SD/C2  FC 1-2 log2(FC) P-val
ETS1-1_rRNA             92.1    96.2    0.2    0.2    1.04    0.06  0.58867
ETS1-2_rRNA             92.1    96.2    0.2    0.2    1.04    0.06  0.58867
ETS2-1_rRNA              0.5     0.6    0.8    0.7    1.39    0.48  0.67094
ETS2-2_rRNA              0.5     0.6    0.8    0.7    1.39    0.48  0.67044
HRA1_ncRNA               1.1     0.5    1.1    0.7    0.45   -1.14  0.61059

[ Output omitted for brevity ]

File                        Features   P < 0.05
fc-3.txt:                       6982       1148
```

```
# To save terminal output to a file:
shell-prompt: ./14c-fasda-fc-hisat2.sh |& tee fasda-fc-hisat2.out

# If the syntax above doesn't work in your shell:
shell-prompt: ./14c-fasda-fc-hisat2.sh 2>&1 | tee fasda-fc-hisat2.out
```

Note
Be sure to carefully review the output to the terminal for warnings, errors, and reasonable values. If you saved a copy of the terminal output by piping through **tee**, you can use tools such as **grep -i error file.out** to search for error messages, etc.

The **fasda fold-change** output, as shown above, is a neatly formatted text file showing the *quantification matrix*, i.e. the mean normalized counts for each condition (MNC1 and MNC2) side-by-side. It also shows the standard deviations of those counts divided by the mean counts (SD/C1 and SD/C2) for easy interpretation as a fraction or percentage, the fold-change (MNC2 / MNC1), and the P-value (probability that a fold-change at least this extreme would occur at random given all possible pairings of the counts for this feature).

Note
The SD/C1 and SD/C2 statistics help us understand why the P-value is what it is. Higher standard deviations in the counts will generally lead to higher P-values (greater likelihood of the null hypothesis that this fold-change arose by chance and does not indicate a significant change in read counts).

After running **fasda fold-change**, the scripts `11b-fasda-fc-kallisto.sh` and `14c-fasda-fc-hisat2.sh` output the total number of features and features with P-values less than 0.05, which is a crude indication of likely *significant differential expressions* (SDEs). We see in the output that the SDE counts for Kallisto (651) and HISAT2 (689) are very similar, as they should be. If there is a drastic difference between these two, we need to go back to the alignment and quantification stages for both Kallisto and Hisat2 and identify the problem.

Note
Be sure to carefully review the output to the terminal for warnings, errors, and reasonable values. If you saved a copy of the terminal output by piping through **tee**, you can use tools such as **grep -i error file.out** to search for error messages, etc.

As always, we should follow the "know your data" principle and look at the output. The scripts `11b-fasda-fc-kallisto.sh` and `14c-fasda-fc-hisat2.sh` display the output using **more** for us, so we can browse it just after running them.

As the first sanity check, we should verify that the **fasda fold-change** output has the same number of features (one per line) as the normalized count files from the same aligner/quantifier. The FC file contains a header line, while the normalized count files do not, so the FC file should show one more line according to the **wc** command.

```
shell-prompt: wc -l Results/11-fasda-kallisto/*
    6584 Results/11-fasda-kallisto/all-norm.tsv
    6584 Results/11-fasda-kallisto/cond1-norm-3.tsv
    6584 Results/11-fasda-kallisto/cond2-norm-3.tsv
    6585 Results/11-fasda-kallisto/fc-3.txt
   20943 total

shell-prompt: wc -l Results/14-fasda-hisat2/*-norm.tsv Results/14-fasda-hisat2/*.txt
    6981 Results/14-fasda-hisat2/all-norm.tsv
    6981 Results/14-fasda-hisat2/cond1-norm.tsv
    6981 Results/14-fasda-hisat2/cond2-norm.tsv
    6982 Results/14-fasda-hisat2/fc-3.txt
   27925 total
```

4.15.2 Comparing results from DE tools

Consider the counts below, for which Sleuth produced a P-value of 0.12, while FASDA produced an exact P-value of 0.024.

```
FASDA:

 Cond1   Cond2  FC   1-2
1165.0  4064.3 3.5 0.023

Sleuth:

    SC1    SC2  SFC    SPV
   70.5  300.4  4.3 0.1203

             R1      R2      R3
Cond1   1045.41 942.707 1220.02
Cond2   2835.7  4167.81 3718.39
```

Given the high fold-change, high read counts, and relatively low variance, this clearly looks like a SDE feature. So why was the Sleuth P-value so high? Tools like Sleuth, DESeq2, and EdgeR use parametric statistical methods aimed at increasing statistical power for low sample sizes while controlling false-positives. These methods are generally helpful, but sometimes backfire. The bottom line is, while the 0.05 rule is a good one mathematically, it is only as reliable as the input data and the statistical model assumptions used to analyze it. Give the results of any differential analysis a generous margin of error, and examine the data more closely for anything near that margin.

My best advice is to compute fold-changes and P-values using multiple tools, and then compare the results. Where the tools disagree, look directly at the read counts for individual replicates. This will help you identify the true SDE features as well as weaknesses in analysis tools. To this end, the DIY-RNA-Seq-DE repository contains scripts **11d-kallisto-deseq2.sh** to generate fold-changes and P-values using DESeq2, and **11e-fasda-deseq2-overlap.sh** to compare the results of FASDA and DESeq2. Results for three replicates are shown below.

To install DESeq2, first install the R statistics package:

```
## FreeBSD/GhostBSD
shell-prompt: pkg install -y R

## Dreckly
# Source dreckly.sh (or dreckly.csh) if you haven't already done so
# in this terminal window.
shell-prompt: source ~/Dreckly/pkg/etc/dreckly.sh

# Install R via Dreckly:
shell-prompt: cd ~/Dreckly/dreckly/math/R
shell-prompt: sbmake install
```

Once you have R installed, follow the instructions for installing DESeq2 at the DESeq2 site: https://bioconductor.org/packages/-release/bioc/html/DESeq2.html.

You will also need to install the R CRAN "tibble" package, which is used to modify the R data structures for subsequent steps:

```
shell-prompt: R
> install.packages("tibble")
> quit()
```

With DESeq2 installed, you're now ready to use the scripts provided in the DIY-RNA-Seq-DE repository:

```
shell-prompt: ./11d-kallisto-deseq2.sh

[ Output omitted for brevity ]

DataFrame with 6584 rows and 6 columns
              baseMean log2FoldChange     lfcSE      stat     pvalue      padj
             <numeric>      <numeric> <numeric> <numeric>  <numeric> <numeric>
YPL071C_mRNA   31.2181      0.0599233  0.354661  0.168959 0.86582877 0.9618239
YLL050C_mRNA  485.1655      0.4423301  0.154681  2.859622 0.00424147 0.0448593
YMR172W_mRNA   74.0322      0.4561252  0.246397  1.851176 0.06414419 0.2990542
YOR185C_mRNA   58.3596     -0.0186674  0.310662 -0.060089 0.95208472 0.9875247
YLL032C_mRNA   28.5098     -0.6651457  0.384567 -1.729594 0.08370277 0.3503916
...               ...            ...       ...       ...        ...       ...
YIL175W        7.00525      -0.635988  0.711149 -0.894310   0.371156  0.733826
YLL016W       48.63727       0.321834  0.312716  1.029156   0.303406  0.683830
YLL017W        5.28864       0.935342  0.832559  1.123455   0.261244  0.636783
YIL174W        0.00000             NA        NA        NA         NA        NA
YIL170W       15.70860       0.423666  0.476531  0.889062   0.373970  0.735527

File                        Features   P < 0.05 Padj < 0.05
kallisto-deseq2-results.tsv:       6585       3167        1193

shell-prompt: ./11e-fasda-deseq2-overlap.sh
Replicates:                       3
Total features:                   6585
SDE features reported by FASDA:   1027
SDE features reported by DESeq2:  1193

Common to FASDA and DESeq2:       747
Reported only by FASDA:           280
Reported only by DESeq2:          446

Overlap = Common-SDE / (Common + FASDA-only + DESeq2-only)
Overlap:                          50%
```

We see here that FASDA and DESeq2 reported similar numbers of SDE features, based on a P-value < 0.05. About 50% of all SDE features reported by either tool were reported by both FASDA (using exact P-values) and DESeq2 (using adjusted P-values). Note that DESeq2 reports *adjusted* P-values to reduce the number of false discoveries that would occur using its raw P-values.

We expect statistical methods to be more reliable with more samples, so not surprisingly, the overlap improves when using more than three replicates:

```
Replicates:                       5
Total features:                   6585
SDE features reported by FASDA:   2234
SDE features reported by DESeq2:  2156

Common to FASDA and DESeq2:       1956
Reported only by FASDA:           278
Reported only by DESeq2:          200

Overlap = Common-SDE / (Common + FASDA-only + DESeq2-only)
Overlap:                          80%
```

```
Replicates:                       10
Total features:                   6585
SDE features reported by FASDA:   2972
SDE features reported by DESeq2:  2744

Common to FASDA and DESeq2:       2600
Reported only by FASDA:           372
Reported only by DESeq2:          144

Overlap = Common-SDE / (Common + FASDA-only + DESeq2-only)
Overlap:                          83%
```

```
Replicates:                       24
Total features:                   6585
SDE features reported by FASDA:   3779
SDE features reported by DESeq2:  3578

Common to FASDA and DESeq2:       3451
Reported only by FASDA:           328
Reported only by DESeq2:          127

Overlap = Common-SDE / (Common + FASDA-only + DESeq2-only)
Overlap:                          88%
```

4.15.3 Interpreting the results

This book is mainly a how-to guide for working through a differential expression analysis, not a research paper analyzing the results. Nevertheless, it is important to understand some of the major issues involved with interpreting results, so we identify and discuss some of them here. Details are left to peer-reviewed research papers on the subject. There are many more opportunities to explore these issues and potentially publish papers of your own on the subject.

The MNC1 and MNC2 values are the mean normalized counts for each condition, i.e. the average count across all replicates for the same condition after applying median ratio normalization.

A *fold-change* is the ratio of mean normalized counts across the two conditions. In theory, a fold-change of exactly 1.0 means there was no change in gene expression. Reality is much more complex. Even if we sample RNA twice *under the same condition*, we will almost never get exactly the same abundance counts. In fact, the counts could vary significantly. They can even vary significantly across technical replicates (partitions of the same biological sample), as described in Section 4.14.

Determining which of the fold-changes are significant, i.e. represent a real biological change, is actually impossible to do with a high degree of confidence, but that's OK. Remember the goal we set in Section 0.3: Discover something of value in the data. Statistical analysis of the fold-changes will help us identify *likely* significant changes, which means we will be less likely to waste time on wild goose chases when we try to experimentally verify the results in the lab. While examining thousands of genes or transcripts, increasing our odds of picking some of the interesting ones will lead to a better use of our time. This is really the only realistic goal for a differential expression analysis.

SD/C1 is the standard deviations of the counts for condition 1, divided by MNC1 to normalize the scale across features with different total counts. Likewise for SD/C2. E.g., if feature 1 has MNCs around 100, while feature 2 has MNCs around 1000, comparing their standard deviations directly would make no sense. A standard deviation of 50 would be very high where the counts are around 100, and very low where the counts are around 1000. A low SD/MNC indicates that the counts were fairly consistent across the replicates. Consistency of counts across biological replicates is largely due to luck, and yet it lowers the P-values (discussed shortly) that we rely on to select features for further study. Hence, a simple metric showing how variable the counts are can be helpful in determining why the P-value is what it is.

The *P-value* is a common statistic used in many analysis within and outside of bioinformatics. It is widely misunderstood and often misinterpreted. What the P-value tells us in this setting, and the only thing it tells us, is the probability that a fold-change *at least* this extreme would occur if all the counts across all the conditions were randomly arranged. A low P-value means that the fold-change we are seeing, or one more extreme, is unlikely to occur by chance. A P-value of 0.05 or lower is traditionally used as a cutoff for rejecting the null hypothesis that "nothing interesting is going on here". It does not mean that "something interesting is definitely going on here", only that there is a low probability that this fold-change arose by dumb luck. This 0.05 standard should not be taken as a hard-and-fast rule, though. The cutoff to use must be chosen intelligently and objectively, based on the particular study.

Note The MNC* and SD/C* statistics reported by FASDA directly impact the P-value. Higher mean normalized counts lead to lower P-values. Lower standard deviations lead to lower P-values.

While higher fold-changes and higher counts lead to higher statistical significance, this does not necessarily correlate to higher biological significance. A fold-change of 1.5 may be highly significant for one gene due to amplification of its effect on the downstream biological pathway, while the same fold-change may be meaningless for another gene. E.g. increasing the abundance of a scarce enzyme by 50% could have a profound effect on a cell, while a 50% increase in many other proteins would have no measurable effect.

Low P-values can also arise with small fold-changes when the variance among counts across replicates is low. Figure 4.33 illustrates this using hypothetical count data analyzed with **fasda pval-sim**.

```
# Low variance
Cond1 Cond2
   24    32
   22    28
   25    30

FC = 1.268  P-value = .033

# High variance
Cond1 Cond2
   19    31
   19    26
   27    24

FC = 1.246  P-value = 0.163
```

Figure 4.33: Effects of low variance

Variance in our read counts is affected by many uncontrollable and unpredictable factors in the cells of different samples, sample prep, and sequencing. When analyzing data from a small number of replicates, low variance will often arise by chance. This can only be discovered by repeating the sequencing experiment or by increasing the number of replicates. Never draw any conclusions based solely on the P-value and/or fold-change.

For a study with three replicates, FASDA actually computes *exact* P-values, by generating every possible arrangement of the six counts (3 replicates * 2 conditions), and computing the fold-changes for each one. The P-value is then simply the number of fold-changes greater than or equal to the one computed as MNC2/MNC1, divided by the total number of possible groupings of three pairs. As the number of possible groupings grows factorially with the number of replicates, computing exact P-values

is only feasible for a small number of replicates. For five to seven replicates, FASDA down-samples the possible groupings to estimate the exact P-value. For eight replicates or more, FASDA uses the Mann-Whitney (A.K.A. Wilcoxon rank-sum) test to estimate the P-value. (8 is the minimum recommended for Mann-Whitney stability.) Other tools such as DESeq2 and edgeR use various other statistical methods to estimate P-values, most of which make assumptions about the data (such as a negative binomial distribution, which does not always fit well).

Caution A low P-value in no way implies that the fold change is likely biologically significant. The mathematical methods used to compute P-values know nothing about biology. They simply take a list of numbers and report the probability of a specific set of arrangements of them.

P-values are highly inaccurate for determining which fold-changes are biologically interesting. They only provide clues that must be corroborated with other evidence before drawing conclusions. Researchers should not assume that a fold-change with a P-value of 0.01 is significant, nor should they rule out one with a fold-change of 0.1. So why do we use them? The answer is simple: We don't have anything better. FASDA throws in SD/MNC in order to help guide the user toward a better selection of features to explore further, but ultimately, it's a bit of an educated guessing game. P-values only give us a hint about which fold-changes are probably worth the effort of investigating. By considering all of these statistics for a given gene together, we come one step closer to deciding whether this change in expression is *biologically* significant. There is no certainty in these data, however. They simply provide the best information that can be gleaned from very limited and uncertain sequence data.

You might be tempted to think that P-values at least tell us which genes had a significant change in activity. However, they don't even correlate well to this metric. What do *gene activity* and *gene expression* mean? Transcription rate? Protein product abundance? Different researchers would choose different definitions based on their goals. However, at the time of this writing, we don't have the tools to easily measure transcription rate or protein abundance. RNA abundance is the only related metric that we can "easily" estimate, where "easy" is defined as prepping a cell sample, extracting and purifying RNA, sending it to a sequencing center, and running various computer programs to interpret the results. It's probably not easy at all by most definitions, but it's the most convenient method we have to date for estimating gene activity. How it reflects transcription rate depends on many factors, such as the degradation rate for the mRNA. Protein abundance depends not only on mRNA abundance, but on other factors that influence the translation rate.

The read counts reported by sequencing studies are highly unreliable due to many natural biological and experimental variables, a small fraction of which are listed below:

- Not every DNA/RNA fragment we extracted actually gets sequenced by the sequencing machine.
- Only part of a fragment is actually sequenced if it is longer than the read length.
- A small change in the timing of mRNA extraction could lead to large changes in abundance.
- Many unknown biological variables influence transcription rate.
- Experimental error could alter mRNA concentration.
- Some genes have multiple *isoforms*, i.e. mRNA transcripts with different start sites, different splicing, etc. Different isoforms serve different functions, and it is not always clear which isoform a particular read came from.

To demonstrate the effects of variability among replicates on a differential analysis, we obtained RNA-Seq data from mouse hippocampus with six biological replicates, and performed a differential analysis on every possible combination of three replicates (6 choose 3 = 20 possible combinations). This comparison shows the effects of variability between biological replicates with other variables held constant. We included a differential analysis for all six samples for comparison.

Three transcripts with reasonably high counts were randomly selected for display here, as low counts tend to be more variable. The data below show the normalized counts (median ratio normalization) for the three transcripts and all biological replicates. We see here that choosing replicates 4, 5, and 6 leads to drastically different counts than choosing replicates 1, 2, and 3, especially where coverage is lower. E.g., for transcript ENSMUST00000000109, the normalized count under condition 1 when using replicates 1, 2, and 3 was 31.2. When using replicates 4, 5, and 6, it was 2553.6.

```
=== ENSMUST00000000090 ===
norm-cond1-1-2-3.tsv       55.0       62.4       54.4
norm-cond1-4-5-6.tsv        0.0       74.7       36.3

=== ENSMUST00000000109 ===
norm-cond1-1-2-3.tsv       36.2       31.2       51.9
norm-cond1-4-5-6.tsv       67.5     2553.6      921.1

=== ENSMUST00000029451 ===
norm-cond1-1-2-3.tsv      652.6     1670.6     1413.9
norm-cond1-4-5-6.tsv      973.6     1118.6     1585.7
```

Spot checking a few counts for two sets of replicates illustrates what *can* happen occasionally, but what about the results for every possible subset? We then ran a standard analysis using FASDA to compute exact P-values for every possible set of three replicates and for all six replicates together. Three sets of three replicates were chosen at random and are shown below. We see from the data below that the mean normalized counts, the fold-changes, and the P-value vary widely depending on which three replicates were chosen. For one of the three transcripts, using a cutoff of $P < 0.05$ would flag the differential expression as significant for some sets of three replicates and not for others, with P-values ranging from 0.02 to 0.91 depending on which replicates we analyzed.

Suppose you were doing the lab work for this study, and decided to do an RNA-Seq differential analysis using three replicates. It is your job to choose six mice (three replicates * two conditions) at random and extract RNA. Depending which mice you choose, transcript ENSMUST00000074945 may or may not be reported as significantly differentially expressed. If we trust the P-value computed when using all six replicates (fc-all.txt, P = 0.81), then the SDE features reported for fc-2-3-4.txt and fc-1-3-4.txt are false discoveries. Fortunately, the odds of selecting mice that lead to this false discovery are low (only two of the 21 combinations analyzed yielded P-values below 0.05).

```
=== ENSMUST00000052368 ===
Replicates        MNC1   MNC2  SD1  SD2    FC   LFC    P
fc-1-3-4.txt     783.6  264.2  0.3  0.6   0.3  -1.6 0.07
fc-1-2-4.txt     734.9  127.8  0.4  1.0   0.2  -2.5 0.08
fc-1-2-3.txt     734.9  286.3  0.4  0.7   0.4  -1.4 0.13
fc-2-4-5.txt     158.6  526.1  1.0  0.3   3.3   1.7 0.13
fc-1-3-6.txt     783.6  458.4  0.3  0.4   0.6  -0.8 0.15
fc-1-2-6.txt     734.9  305.5  0.4  0.8   0.4  -1.3 0.18
fc-2-4-6.txt     158.6  442.3  1.0  0.4   2.8   1.5 0.19
fc-2-5-6.txt     296.1  602.4  0.7  0.4   2.0   1.0 0.24
fc-4-5-6.txt     313.4  610.0  0.6  0.5   1.9   1.0 0.25
fc-1-3-5.txt     783.6  548.9  0.3  0.3   0.7  -0.5 0.26
fc-1-2-5.txt     734.9  394.3  0.4  0.7   0.5  -0.9 0.27
fc-2-3-5.txt     330.4  541.8  0.7  0.4   1.6   0.7 0.32
fc-3-4-5.txt     356.2  535.8  0.6  0.3   1.5   0.6 0.36
fc-2-3-6.txt     330.4  452.6  0.7  0.4   1.4   0.5 0.52
fc-3-5-6.txt     493.7  614.0  0.2  0.4   1.2   0.3 0.52
fc-1-5-6.txt     749.3  610.8  0.4  0.4   0.8  -0.3 0.58
fc-1-4-6.txt     611.8  449.6  0.6  0.4   0.7  -0.4 0.60
fc-3-4-6.txt     356.2  448.5  0.6  0.4   1.3   0.3 0.61
fc-2-3-4.txt     330.4  260.0  0.7  0.6   0.8  -0.3 0.70
fc-1-4-5.txt     611.8  535.4  0.6  0.3   0.9  -0.2 0.82
fc-all.txt       464.9  503.7  0.4  0.7   1.1   0.1 0.82

=== ENSMUST00000074945 ===
Replicates        MNC1   MNC2  SD1  SD2    FC   LFC    P
fc-2-3-4.txt     556.0  300.6  0.2  0.2   0.5  -0.9 0.02
fc-1-3-4.txt     542.8  305.4  0.2  0.2   0.6  -0.8 0.03
fc-2-3-6.txt     556.0  402.8  0.2  0.2   0.7  -0.5 0.06
fc-1-3-6.txt     542.8  407.6  0.2  0.2   0.8  -0.4 0.08
fc-1-2-4.txt     502.6  364.1  0.2  0.3   0.7  -0.5 0.14
fc-2-3-5.txt     556.0  431.8  0.2  0.3   0.8  -0.4 0.14
fc-2-4-6.txt     482.4  392.1  0.2  0.2   0.8  -0.3 0.18
fc-1-3-5.txt     542.8  437.6  0.2  0.3   0.8  -0.3 0.21
```

```
fc-4-5-6.txt      394.1  499.3  0.2  0.3   1.3   0.3 0.26
fc-1-4-6.txt      469.2  398.4  0.2  0.3   0.8  -0.2 0.29
fc-2-4-5.txt      482.4  418.9  0.2  0.3   0.9  -0.2 0.41
fc-1-2-6.txt      502.6  469.0  0.2  0.2   0.9  -0.1 0.48
fc-3-4-6.txt      451.9  397.6  0.3  0.3   0.9  -0.2 0.53
fc-1-2-3.txt      502.6  452.3  0.2  0.3   0.9  -0.1 0.54
fc-1-4-5.txt      469.2  426.5  0.2  0.3   0.9  -0.1 0.57
fc-3-5-6.txt      475.2  504.5  0.3  0.3   1.1   0.1 0.77
fc-3-4-5.txt      451.9  426.7  0.3  0.3   0.9  -0.1 0.78
fc-all.txt        458.0  471.8  0.2  0.2   1.0   0.0 0.81
fc-1-5-6.txt      492.5  501.6  0.2  0.3   1.0   0.0 0.85
fc-2-5-6.txt      505.8  495.3  0.2  0.3   1.0  -0.0 0.88
fc-1-2-5.txt      502.6  497.5  0.2  0.2   1.0  -0.0 0.91

=== ENSMUST00000203177 ===
Replicates         MNC1   MNC2  SD1  SD2    FC   LFC    P
fc-1-4-5.txt      164.7  204.5  0.3  0.2   1.2   0.3 0.11
fc-2-5-6.txt      229.9  192.5  0.2  0.2   0.8  -0.3 0.14
fc-1-3-5.txt      109.6  216.9  0.7  0.2   2.0   1.0 0.15
fc-1-3-6.txt      109.6  201.4  0.7  0.2   1.8   0.9 0.19
fc-1-2-5.txt      178.1  211.4  0.3  0.2   1.2   0.2 0.21
fc-1-4-6.txt      164.7  189.7  0.3  0.2   1.1   0.2 0.25
fc-1-3-4.txt      109.6  174.2  0.7  0.3   1.6   0.7 0.30
fc-3-4-5.txt      134.7  204.5  0.7  0.2   1.5   0.6 0.34
fc-all.txt        163.0  195.0  0.4  0.2   1.2   0.3 0.40
fc-2-3-5.txt      146.7  213.7  0.7  0.2   1.5   0.5 0.43
fc-3-4-6.txt      134.7  189.3  0.7  0.2   1.4   0.5 0.45
fc-2-4-6.txt      201.9  186.9  0.2  0.2   0.9  -0.1 0.49
fc-1-2-6.txt      178.1  196.6  0.3  0.2   1.1   0.1 0.51
fc-2-3-6.txt      146.7  198.8  0.7  0.2   1.4   0.4 0.51
fc-1-2-3.txt      178.1  140.0  0.3  0.7   0.8  -0.3 0.58
fc-3-5-6.txt      162.7  196.0  0.6  0.2   1.2   0.3 0.64
fc-4-5-6.txt      206.4  193.7  0.3  0.3   0.9  -0.1 0.66
fc-2-3-4.txt      146.7  171.5  0.7  0.3   1.2   0.2 0.71
fc-1-2-4.txt      178.1  170.6  0.3  0.3   1.0  -0.1 0.79
fc-1-5-6.txt      192.8  194.8  0.3  0.2   1.0   0.0 0.93
fc-2-4-5.txt      201.9  200.7  0.2  0.2   1.0  -0.0 0.96
```

The bottom line is, don't take fold-changes and P-values too seriously. We have to use *something* to narrow down the list of genes to investigate, and we don't have anything better to go on until we do more lab work. As mentioned earlier, the statistics here are based on questionable inputs, and only serve to improve our odds of picking interesting genes. Being good at differential expression analysis is like being good at poker. Most of the game is out of our control, but if we understand how it's played, we can improve our odds of winning.

In any differential expression analysis, due to random choices you make and a slew of other variables beyond your control, you *will* overlook some DEGs (differentially expressed genes), you *will* waste time on some false discoveries, and you *will* discover something of value.

4.15.4 How many replicates?

One general rule of thumb in statistics is to use a sample size of 30 or more whenever possible, to ensure high confidence in the results. For RNA-Seq experiments, this is typically far out of reach due to the cost of preparing and sequencing so many samples. Typical RNA-Seq experiments use only three replicates. According to Schurch, et al [Schurch], with such a small sample size, many SDE features will be missed:

> With three biological replicates, nine of the 11 tools evaluated found only 20% to 40% of the SDE features identified with the full set of 42 clean replicates.

We *will* miss things when using such a small sample size, because small sample sizes generally mean low statistical power (the ability to identify true positives / avoid false negatives). This is not ideal, but it's good enough for our purposes. Remember

our primary goal from Section 0.3: Discover something of value in the data. The truth is, even though many SDE features will be missed, there is generally *a lot* to be discovered in differential analysis results, and we will discover some of it, even with a meager sample size of three replicates.

The bigger concern for biologists is avoiding *false positives*, where the statistics lead us to believe that something is significant, when it really is not. Put mathematically, a false positive, or *false discovery*, is where we reject the null hypothesis that "nothing interesting is happening here", when should have accepted it. This is a big problem, because experimentally verifying results in the lab can be time-consuming and costly. We want to expend our limited resources on investigations that are likely to yield results. Hence, limiting the *false discovery rate* (FDR) is one of the most important capabilities of a differential analysis tool. This sometimes comes at the expense of being able to detect true positives. FASDA computes exact P-values for experiments with fewer than eight replicates, and Mann-Whitney P-values for eight or more. Both of these methods are known to minimize FDR.

As the cost of sequencing continues to decline, we may see an increase in the number of replicates used in a typical RNA-Seq experiment, leading to more reliable information on fold-changes and P-values. However, *this will not change our basic strategy*. The numbers reported by differential analysis tools will better reflect the data we have, but remember that the data have not improved, we just have more of it. Increasing the number of replicates does not change the fact that the counts we feed into the statistical tools are unreliable, for all the reasons stated in Section 4.15.3. We still need to take the statistics with grain of salt and employ other factors in decisions about how to spend precious time and other resources.

Increasing the number of replicates generally decreases P-values, as a higher sample count increases statistical power. Hence, as the sample counts increase, we may want to decrease the cutoff P-value, while still considering other statistical and biological factors before deciding if a change is significant. Figure 4.34 shows the effect of adding more replicates from the same yeast data on P-values for one particular transcript. With a P-value of 0.44 for three replicates, we would never likely consider this a SDE feature. However, with six or more replicates, we are led to believe this gene warrants further investigation. The pattern is similar for most other transcripts.

```
Results/11-fasda-kallisto/fc-3.txt
Feature                 MNC1    MNC2  SD/C1  SD/C2  FC 1-2 log2(FC) P-val
YMR267W_mRNA            53.3    44.9    0.3    0.2    0.84   -0.25  0.44433

Results/11-fasda-kallisto/fc-4.txt
Feature                 MNC1    MNC2  SD/C1  SD/C2  FC 1-2 log2(FC) P-val
YMR267W_mRNA            58.7    47.9    0.3    0.2    0.82   -0.29  0.23655

Results/11-fasda-kallisto/fc-5.txt
Feature                 MNC1    MNC2  SD/C1  SD/C2  FC 1-2 log2(FC) P-val
YMR267W_mRNA            57.2    46.8    0.3    0.2    0.82   -0.29  0.13969

Results/11-fasda-kallisto/fc-6.txt
Feature                 MNC1    MNC2  SD/C1  SD/C2  FC 1-2 log2(FC) P-val
YMR267W_mRNA            63.0    46.2    0.2    0.2    0.73   -0.45  0.04512

Results/11-fasda-kallisto/fc-7.txt
Feature                 MNC1    MNC2  SD/C1  SD/C2  FC 1-2 log2(FC) P-val
YMR267W_mRNA            62.8    47.2    0.2    0.2    0.75   -0.41  0.03527

Mann-Whitney P-values from here on:

Results/11-fasda-kallisto/fc-8.txt
Feature                 MNC1    MNC2  SD/C1  SD/C2  FC 1-2 log2(FC) P-val
YMR267W_mRNA            60.4    47.5    0.2    0.2    0.79   -0.35  0.05871

Results/11-fasda-kallisto/fc-9.txt
Feature                 MNC1    MNC2  SD/C1  SD/C2  FC 1-2 log2(FC) P-val
YMR267W_mRNA            60.4    48.7    0.2    0.2    0.81   -0.31  0.03051

Results/11-fasda-kallisto/fc-10.txt
Feature                 MNC1    MNC2  SD/C1  SD/C2  FC 1-2 log2(FC) P-val
YMR267W_mRNA            59.5    47.7    0.2    0.2    0.80   -0.32  0.01556
```

Figure 4.34: Effect of increasing replicates on P-values

A direct effect of the decreasing P-values that arise from using more replicates is that more DE features will be flagged as significant, assuming we use the same cutoff. Below are data that show the number of DE features with P-values under 0.05 for a variety of replicate counts from 3 to 48.

```
Exact or near-exact P-values:

File                        Features   P < 0.05
fc-3.txt                        6585       1035

File                        Features   P < 0.05
fc-4.txt:                       6585       1823

File                        Features   P < 0.05
fc-5.txt:                       6585       2231

File                        Features   P < 0.05
fc-6.txt:                       6585       1974

File                        Features   P < 0.05
fc-7.txt:                       6585       2358

Mann-Whitney from here on:

File                        Features   P < 0.05
```

```
fc-8.txt:                       6585       2719

File                        Features   P < 0.05
fc-9.txt:                       6585       2876

File                        Features   P < 0.05
fc-10.txt:                      6585       2969

File                        Features   P < 0.05
fc-24.txt:                      6585       3778

File                        Features   P < 0.05
fc-48.txt:                      6585       4417
```

We see here that using a high number of replicates, more than half of the transcripts in the transcriptome are identified as significantly differentially expressed. This certainly alleviates our worries about false negatives (missing things that may be worthy of further study), but introduces a new problem: We now have thousands of SDE features to experimentally verify, unless we find another way to narrow the list.

Note that in the data above above that there was actually a *drop* in the number of DE features with a P-value under 0.05 when we added the sixth replicate. As this is unexpected, we repeated this experiment using a different set of samples, starting with biological replicate 7 instead of 1. The results below show a steady increase in SDE features with increased number of replicates until the 7th replicate (biological replicate 13 from the study). This suggests that the data in replicates 6 and 13 is of questionable quality. Indeed, [Schurch] identified 6 of the 48 biological replicates of the wild-type yeast, and four of the SNF2 mutant replicates used here as low quality.

```
Exact or near-exact P-values:

File                        Features   P < 0.05
fc-3.txt:                       6585       1306

File                        Features   P < 0.05
fc-4.txt:                       6585       1797

File                        Features   P < 0.05
fc-5.txt:                       6585       2104

File                        Features   P < 0.05
fc-6.txt:                       6585       2535

File                        Features   P < 0.05
fc-7.txt:                       6585       2513

Mann-Whitney from here on:

File                        Features   P < 0.05
fc-8.txt:                       6585       2609

File                        Features   P < 0.05
fc-9.txt:                       6585       2659

File                        Features   P < 0.05
fc-10.txt:                      6585       2918
```

4.15.5 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. The file `normalized.tsv` contains counts for condition 1 in columns 2, 4, and 6, and condition 2 in columns 3, 5, and 7. Column 1 contains the feature names. Show the Unix command to create the files `cond1.tsv` and `cond2.tsv`.
2. What is the first sanity check we should perform on **fasda fold-change** output?
3. What is a fold-change?
4. What is a P-value in terms of fold-changes?
5. How reliable are P-values for determining which genes have seen a significant change in expression? What is one way to get a better idea about which DE features to focus on?
6. What typically happens when we sample RNA more than once under the exact same conditions?
7. How do P-values help us cope with unreliable abundance estimates? Is the situation just hopeless?
8. How to standard deviations in the counts help us understand the results of a DE analysis?
9. What do P-values tell us about the biological significance of a fold-change?
10. How is an exact P-value computed for a fold-change? Why don't we always use them?
11. Why are read counts so unreliable? Keep your answer simple and general.
12. As a biologist, are we more concerned about false negatives (missing something significant) or false positives (exploring something that is actually not significant)?
13. What are the typical effects of increasing the number of replicates?

4.16 Visualizing DE results

It is often helpful to visualize results graphically. For differential expression data, the conventional visualization tool is the *heatmap*, where the counts are represented as color intensities, as shown in Figure 4.35.

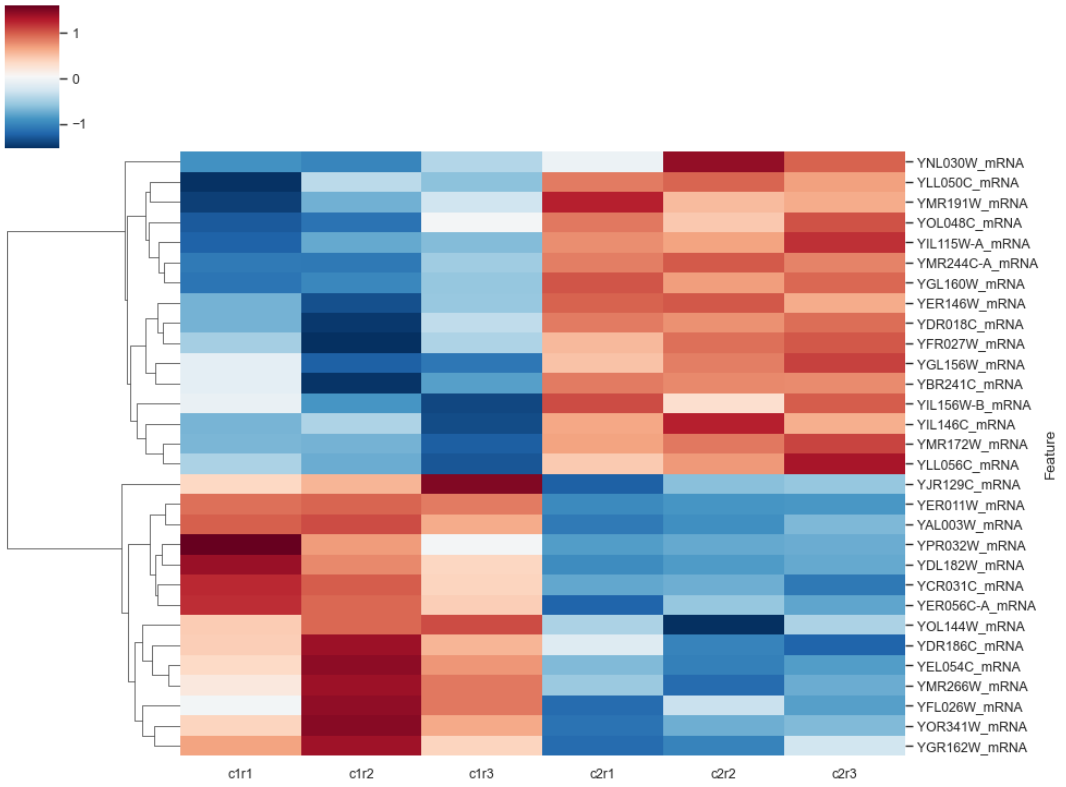

Figure 4.35: Clustered heatmap

In this heatmap, red represents higher counts, while blue represents lower counts. A shift from blue to red going from left to right means condition 2 (SNF2 mutant) was up-regulated relative to condition 1 (wild-type). The dendrogram on the left shows how the features cluster according to the pattern in their read counts, i.e. it groups features according to how closely their count profiles correlate. This information can be useful in determining which genes are *coregulated*, which might indicate a relationship to each other within a biological pathway. Or, it might just be a coincidence. That's science.

Note that the colors represent counts that are transformed to produce a more meaningful visualization. If we used the actual read counts, then a feature with read counts in the thousands would be represented by very different colors than one that had read counts in the dozens. A common approach in RNA-Seq DE heatmaps is to use the *Z-scores* of the counts, i.e. the number of standard deviations away from the mean. By using Z-scores, we effectively normalize the colors across all features, so that the color ranges can be compared across features. I.e., an intense red means a high read count *for that feature*, regardless of how the expression level compares to other features. We must be aware of these types of transformations when interpreting any visualization, which is almost always a simplification (a.k.a. *reduction*) of the actual data.

We visualize the individual read counts for all replicates, so that we also can see the variability within each condition as well as the change across conditions. Like the SD/C1 and SD/C2 columns in the **fasda fold-change** output, this helps us determine why the P-value is what it is. It can also help identify problems with the raw data. Figure 4.36 shows a heatmap for 10 replicates. We can see here that sample "c2r6" (condition 2 = SNF2 mutant, replicate 6) is an outlier, i.e., the colors do not align with the other condition 2 samples shown. This corroborates our suspicion about the drop in SDE features we saw in in Section 4.15.4 when adding the 6th replicate. Recall that we generally expect to see an increase in SDE features when adding more replicates, but adding the 6th replicate of Yeast data caused a decrease.

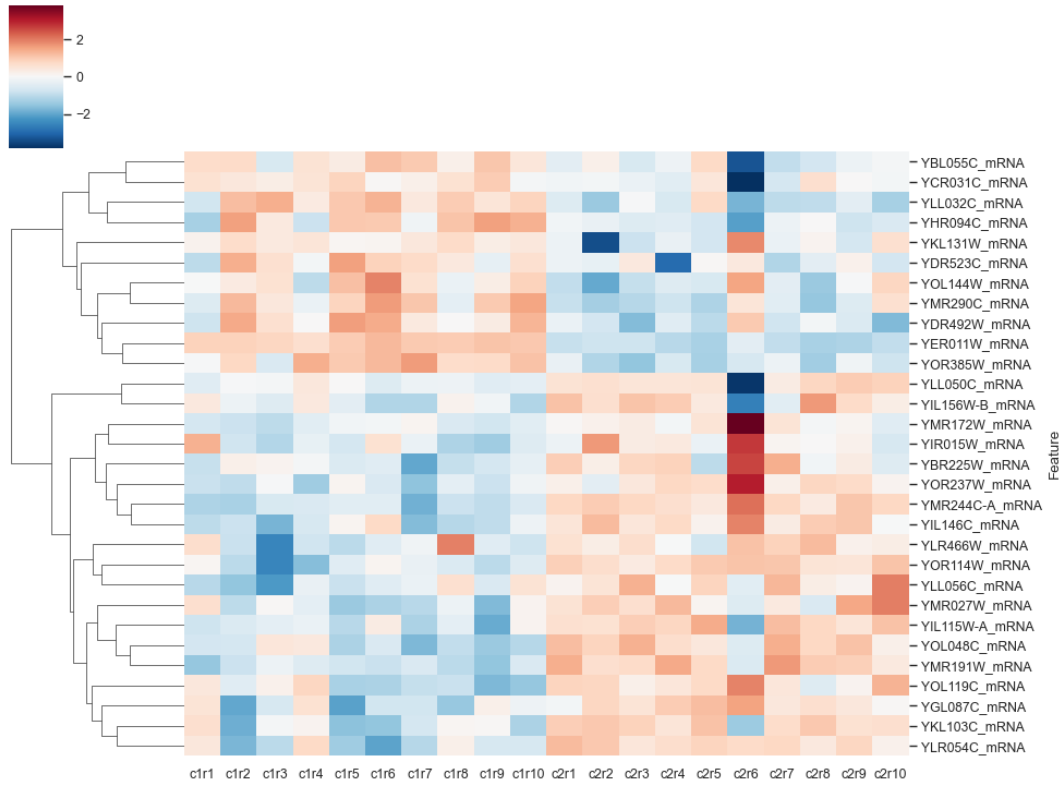

Figure 4.36: Clustered heatmap

There are several tools available for generating heatmaps, and every bioinformatician has their own preference. The R language offers several methods including the basic `heatmap()` function, the ggplot2 package and the Plotly R package. There are Python libraries such as the generalized matplotlib, the statistics-focused seaborn, and the interactive plotly. Finally, there is the tried and true gnuplot (which has nothing to do with the GNU software project, the name is coincidental). Gnuplot has been under active development since 1986, and supports both 2D and 3D plotting. I chose seaborn over the others for the following reasons:

- Gnuplot, while very powerful and mature, does not currently support *clustering* (dendrograms to group of features with similar expression patterns, i.e. up- or down-regulation).
- R, while a very powerful and convenient tool for interactively doing statistical analyses, is problematic for creating applications that are easy to install and use. Often, the only convenient way to install the R packages needed for an R script is by doing it manually in the R environment before running the script. Ironically, installing R packages from within an R script or from the Unix command-line is more difficult than installing them manually from the interactive R environment. Some R packages do exist in other package managers such as FreeBSD ports and dreckly, making automated installation much easier, but there is less motivation to create and maintain these packages because the R community puts most of their effort into installing things from withing the R environment.
- Seaborn is a python library, easily installed via most common package managers, including FreeBSD ports and dreckly. Hence, we have simply included the seaborn package as a dependency for the fasda-utils FreeBSD port and dreckly package, which provides a Python heatmap script that just works out of the box.

Generating a heatmap is usually one of the steps that involves programming when using traditional tools. For the pipeline presented in this text, you can view a heatmap based on FASDA output by simply creating a text file with a list of features, and running one simple command. Creating the text file will involve some manual investigation to identify the features (genes or transcripts) that you consider most interesting. However, it does not involve any programming. First, use **fasda filter** to select from the list of features produced by **fasda fold-change**:

```
# Count total features in the FASDA output
shell-prompt: wc -l Results/14-fasda-hisat2/fc-3.txt
    6982 Results/14-fasda-hisat2/fc-3.txt

# Narrow the list by selecting for low P-values and high fold-change
shell-prompt: fasda filter --max-p-val 0.05 --min-fc 2 \
    Results/14-fasda-hisat2/fc-3.txt > sdegs.txt
shell-prompt: wc -l sdegs.txt
     470 sdegs.txt

# Show the header line describing the features, since it's not in sdegs.txt
shell-prompt: head -n 1 Results/14-fasda-hisat2/fc-3.txt
Feature                 MNC1    MNC2  SD/C1  SD/C2  FC 1-2 log2(FC) P-val

# Show the first 5 lines of sdegs.txt
shell-prompt: head -n 5 sdegs.txt
RDN5-4_rRNA              5.8     2.0    0.4    0.3    0.35   -1.50  0.04680
RDN58-1_rRNA           355.2    81.8    0.5    0.2    0.23   -2.12  0.04384
RDN58-2_rRNA           358.5    84.9    0.5    0.2    0.24   -2.08  0.04384
RPR1_ncRNA               7.3     2.0    0.3    0.3    0.27   -1.87  0.02562
YAL016C-A_mRNA          10.4     4.9    0.2    0.3    0.47   -1.07  0.03645
```

We see above that selecting features with a P-value <= 0.05 and fold-change >= 2 narrowed the list from 6,982 features to just 470. This is still a large list, but much less daunting. You can play around with other P-values and fold-changes, as well as other parameters. Just run **fasda filter** with no more arguments for a quick explanation of the options, or run **man fasda-filter**.

```
shell-prompt: fasda filter

Usage: fasda filter [--min-mnc N] [--max-sd N] [--min-fc N] [--max-p-val N] fc-file.txt

* At least one filtering flag above must be specified.

* Average of MNC1 and MNC2 must be >= min-mnc       Default = 1
* Average of SD/C1 and SD/C2 must be <= max-sd      Default = 1.0
* FC or 1/FC must be >= min-fc                      Default = 1.0
* P-val must be <= max-p-val                        Default = 0.05

Example data:

Feature                 MNC1    MNC2  SD/C1  SD/C2  FC 1-2 log2(FC) P-val
YPL071C_mRNA            29.6    31.1    0.3    0.2    1.05    0.07  0.72611
YLL050C_mRNA           399.5   543.8    0.2    0.2    1.36    0.44  0.02857
YMR172W_mRNA            60.6    83.3    0.2    0.2    1.38    0.46  0.02167
YOR185C_mRNA            57.1    56.1    0.3    0.2    0.98   -0.03  0.92562
YLL032C_mRNA            33.9    21.6    0.3    0.3    0.64   -0.65  0.14138
YBR225W_mRNA            61.4    75.4    0.2    0.2    1.23    0.30  0.11281

Example command:

/usr/local/libexec/fasda/filter --min-mnc 20 --max-sd 0.3 --min-fc 2.0 --max-p-val 0.05  ↩
    Results/11-fasda-kallisto/fc-3-replicates.txt

shell-prompt: man fasda-filter
```

Then do some investigation to identify genes/transcripts of interest to your study, possibly using some of the functional genomics tools described in Section 4.19, which help us identify gene functions and interactions. After selecting your features of interest, create a simple text file, e.g. `features-of-interest.txt`, containing the feature names in column 1. It need not contain any other columns, but it can. Then run **fasda heatmap**, with your short features file as the first argument, and the outputs of **fasda normalize** as the second and third:

```
# Generate a short list of features.  Here we cheat and just take the first
```

```
# 30 SDEGs from sdegs.txt.  In reality, you will want to select genes of
# interest based on their behavior here and known functions/interactions.
shell-prompt: head -n 30 sdegs.txt > features-of-interest.txt

# Generate and display a heatmap of the counts
shell-prompt: fasda heatmap features-of-interest.txt \
    Results/14-fasda-hisat2/cond1-norm.tsv \
    Results/14-fasda-hisat2/cond2-norm.tsv
```

Note
The **fasda heatmap** script is installed by the FreeBSD port and dreckly package `biology/fasda-utils`, a collection of optional companion programs and scripts for FASDA. They are kept separate in order to limit the size and build time of the core FASDA packages, and not included in the rna-seq meta-packages for the same reason. This is because the Python seaborn package has a massive list of dependencies that can take hours to build on a typical PC.
The **fasda heatmap** command requires access to a graphical display. It may or may not work when running on a remote Unix system over an **ssh** connection. See The Research Computing User's Guide for information about enabling remote graphical applications.

The DIY-RNA-Seq-DE repository contains two simple example scripts for displaying heatmaps of the sample Yeast data. Figure 4.35 shows the output of the **11c-fasda-kallisto-heatmap.sh** script. This script filters the FASDA Kallisto outputs for fold-changes with low P-values, then generates a heatmap for the read counts of the first 30 features selected, which of course is a meaningless selection, but it serves to demonstrate how a heatmap works. In reality, you will want to do a careful selection of features to visualize. The **14d-fasda-hisat2-heatmap.sh** does the same for the hisat2 results, and should produce a heatmap that looks very similar.

4.16.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

---

1. How are read counts represented in a heatmap?
2. What does the dendrogram next to the heatmap tell us?
3. Why does a heatmap show read counts for all samples, rather than just the mean normalized count for each condition?
4. Which plotting library is the best?
5. How do we quickly and easily select the interesting SDE features from the list produces by **fasda fold0-change**?

4.17 Transcript or gene level expression analysis

The examples in previous sections show a transcript-level differential expression analysis. This type of analysis preserves the most possible information about the expression patterns in our data. Remember that the RNA-Seq protocol sequences *RNA* transcripts, not genes.

Gene-level analyses can be a little problematic, since genes may produce multiple isoforms (different transcripts based on alternative start sites and/or alternative splicing). The different isoform transcripts may serve very different functions within the cell, so lumping them all together into one "gene expression level" would be nonsensical in some cases, and discards information in all cases.

If a gene-level analysis is desired, we can easily translate from transcripts to genes, provided that we have a feature map such as a GTF or GFF3 file. These files indicate the *parent feature* of every feature, so it is easy to identify the gene from which each transcript was produced. The biolibc-tools package, installed by the rna-seq package, contains a tool for this purpose that outputs gene names associated with each feature, given a GFF3 file and a file with a list of features. One way to derive a feature list is by extracting the names from an abundance file.

```
# Check out the format of an abundance file
shell-prompt: head Results/14-fasda-hisat2/sample01-cond1-rep01-trimmed-abundance.tsv
target_id       length  eff_length      est_counts      tpm     fpkm
ETS1-1_rRNA     700     0.0     56.2    73.994179       82.712845
ETS1-2_rRNA     700     0.0     56.2    73.994179       82.712845
ETS2-1_rRNA     211     0.0     0.0     0.000000        0.000000
ETS2-2_rRNA     211     0.0     0.0     0.000000        0.000000
HRA1_ncRNA      564     0.0     2.1     3.423575        3.826972
ICR1_ncRNA      3199    0.0     20.4    5.893961        6.588441
IRT1_ncRNA      1489    0.0     16.6    10.259662       11.468548
ITS1-1_rRNA     361     0.0     9.7     24.872496       27.803196
ITS1-2_rRNA     361     0.0     10.3    26.291975       29.389935

# Extract column 1 and filter for mRNA features
shell-prompt: cut -f 1 Results/14-fasda-hisat2/sample01-cond1-rep01-trimmed-abundance.tsv | ↩
     grep mRNA > features.txt

shell-prompt: head feature-names.txt
YAL001C_mRNA
YAL002W_mRNA
YAL003W_mRNA
YAL004W_mRNA
YAL005C_mRNA
YAL007C_mRNA
YAL008W_mRNA
YAL009W_mRNA
YAL010C_mRNA
YAL011W_mRNA

shell-prompt: blt ensemblid2gene \
    Results/08-reference/Saccharomyces_cerevisiae.R64-1-1.106.gff3 \
    feature-names.txt
YAL064C-A_mRNA  TDA8
YAL061W_mRNA    BDH2
YAL053W_mRNA    FLC2
YAL047C_mRNA    SPC72
YAL016C-B_mRNA  unnamed
YAL015C_mRNA    NTG1
YAR007C_mRNA    RFA1
YAR035W_mRNA    YAT1

[ Output omitted for brevity ]
```

There are also numerous web-based tools and tools for use within scripting languages such as R, Perl and Python.

4.17.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. Why is gene-level differential analysis problematic?
2. How do we convert from transcript IDs to gene IDs (or names)?

4.18 Multiple conditions

When analyzing data with more than two conditions, we simply compare all possible combinations of two conditions individually. For example, if the data represent three time points in development, T1, T2, and T3, then we perform the following comparisons:

```
T1 vs T2
T1 vs T3
T2 vs T3
```

If we discover a statistically significant change in any one of these three comparisons, then we have a significant change overall. We also have more detailed information about when and how fast the change occurred.

4.18.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. List the comparisons that must be performed for a differential analysis using three conditions, "wild-type", "mutant1", "mutant2", and "mutant3".

4.19 Functional genomics

Now that we have identified at least some of the SDE features, it's time to start exploring what they mean. What do these DE genes or transcripts do? Do they control development and regeneration? Are they part of the immune system? Do they have multiple functions? What other genes or transcripts are they known to interact with?

The computational portion of the analysis, which is the primary focus of this text, is now mostly behind us. This is where the expressway ends, and we have to get out and walk the rest of the way. Exploring gene/transcript functions is not a primarily computational process. It involves manual labor carried out by a biologist who understands the biology behind the data. There are many interactive tools and websites containing functional data. They will help us discover the functions of SDE genes, as well as point us to known interactions with other genes. Now that you have a list of differentially expressed features in-hand, we'll point you in the right direction for the next step of your research and then let go of your handlebars, so to speak.

Current knowledge of gene/transcript function and interaction is very limited, because determining and verifying it is a lot of manual labor. If you don't believe me, look at a list of PhD thesis titles from any biology department. People often spend years studying a small group of genes before publishing a paper about them.

The tools for functional analysis are also few and have many limitations. As such, there are not many comparative papers on the subject [Gasperskaja]. We'll point you to a few of the most commonly used tools here, and leave the details to their documentation. If we try to explain them in detail here, they will probably have changed by the time you read this, so this text will be outdated.

KEGG (Kyoto Encyclopedia of Genes and Genomes, according to their website, "is a database resource for understanding high-level functions and utilities of the biological system, such as the cell, the organism and the ecosystem, from molecular-level information, especially large-scale molecular datasets generated by genome sequencing and other high-throughput experimental technologies". It contains, among other things, a collection of manually drawn pathway maps describing molecular interactions, and links between genes and protein products.

DAVID (Database for Annotation, Visualization, and Integrated Discovery), is an online resource that helps biologists analyze large lists of genes. It provides functional annotations and *enrichment analysis* tools. Enrichment analysis is a statistical method of identifying *overrepresented* genes among a large list. Hence, it can be a useful tool for identifying genes of interest out of the thousands that a tool such as FASDA flags as significantly differentially expressed.

GO (Gene Ontology) is a web-based resource for information on the functions of genes. It defines a standardized terminology, called *GO terms*, that we can use to search for information about:

- Cellular components
- Molecular function of gene products
- Biological processes (sequences of molecular events)

These tools and databases are rapidly changing, so the best source of information about them is their own documentation.

4.19.1 Practice

Note Be sure to thoroughly review the instructions in Section 0.6 before doing the practice problems below.

1. What is the goal of functional analysis following DE analysis?
2. Are there many computational tools to automatically perform a functional analysis?

Chapter 5

Other computing resources

5.1 RNA-Seq analysis on HPC

High Performance Computing, commonly abbreviated *HPC*, is a system for running many processes *in parallel* (at the same time), as opposed to *serially* (one after another). Most personal computers now have multiple processors, and hence can do parallel computing. However, many scientific computations would still take too long to complete using just one computer with four or eight processors. Some examples include weather forecasting and de novo genome assemblies.

HPC refers to the use of *clusters* of anywhere from a few to thousands of computers (called compute nodes) working together. Part II of The Research Computing User's Guide covers the use of HPC clusters in detail.

HPC clusters require the use of a *batch system*, which keeps track of busy and available resources (processors and memory, mainly), and schedules jobs according to a policy such as first-in, first-out.

The most common scheduler used on large research clusters is Slurm. A new scheduler, which is much simpler, easy to set up, and easy to use, is LPJS. With LPJS, you can quickly and easily turn your Mac or other Unix computer into a one-node cluster, so you can queue jobs to run in the background as you would on a large cluster. LPJS also supports clusters with any number of nodes, as well as limiting resource use to a portion of a node's memory and processors. The Research Computing User's Guide contains tutorials for LPJS, Slurm, and a few other popular batch systems.

Fortunately, most RNA-Seq differential analyses do not require a cluster, and can be completed in a few days using one reasonably powerful PC. In fact, most bioinformatics work does not require a cluster. Use of a cluster may be of interest in some cases, though. For example, mapping reads to a large and complex genome may take days on a single PC, but only a few hours on a cluster.

Lastly, you might just want to use a job scheduler to run your analysis steps, rather than run them manually. This helps keep things organized and well-documented, for one thing. It also allows you to submit multiple jobs at once, even if you don't have the computing resources for them to run at the same time. Some of them will just wait in a queue and start running when the resources become available, quite possibly while you're sleeping, or focused on other tasks. The ability to queue jobs like this could knock a few days off your analysis by keeping the computer working when you are not.

If you're interested in learning more about HPC clusters, we recommend setting up a one-node instant cluster on your Unix computer using LPJS. Doing so does not require administrator rights, so you could even try it out on a machine managed by your I.T. department.

Some examples for running RNA-Seq differential expression analysis under LPJS and Slurm can be found at https://github.com/-auerlab/fasda/tree/main/Yeast-test/LPJS, https://github.com/auerlab/CNC-EMDiff/tree/master/RNA-Seq, https://github.com/auerlab/-Optic-Regen/tree/main/RNA-Seq, and https://github.com/auerlab/AxoXenONReg.

5.2 RNA-Seq in the cloud

The term *cloud computing* has a nebulous (pun intended) ring to it, which makes it an excellent marketing term. In reality, the cloud is simply a pool of remote servers that people can allocate, configure, and use as they see fit. There are many cloud services

available today, including those from big players such as Amazon EC2, Google Cloud Platform (GCP), Microsoft Azure. There are also many smaller services that you may find easier to use, such as https://mnx.io/.

The key advantage of cloud services for many people is the ability to pay as you go. I.e., you don't have to purchase and maintain your own computer. Instead, you configure a real computer or a virtual machine, called a *cloud instance* in the cloud service, and only pay for what you use. You can configure an instance with as few or as many processors as you like, around 64) and as little or as much memory as you like, up to the limits of the available hardware. The hourly cost of the cloud instance is proportional to disk-space + total-memory * processors * time-running. I.e., you pay a small rate for disk space occupied whether or not the instance is running, and a separate rate depending on the amount of memory and the number of processors the instance uses while it is running. You still have to manage the operating system and software, just as you would on a real computer, but the company running the cloud service maintains all the hardware.

The payment model may be more cost-effective than owning your own hardware, particularly if your own hardware would not see regular use. So, you might do most of your work on a low-end PC and spin up a cloud instance for the rare occasions that you need more computing power. If, on the other hand, you're running analyses 24/7/365.26, then the rental costs will add up and you may be better off buying a computer.

One potential problem with cloud computing is the remoteness of the servers. Even the biggest players in cloud services will only have a few data centers, and the nearest one may be 1,000 miles or more from your location. This means the Internet connection to it will be slow, and transferring large sequence files to and from it will take a long time, unless you use a commercial high-speed file transfer tool such as Globus. Using remote servers also means you won't have the same easy access to a graphical desktop environment, which you will need for visualizing your FastQC results, heatmaps, etc. If you don't mind transferring files back to your PC for this purpose, then this shouldn't be a problem.

Lastly, the funding model used by many research institutions makes pay as you go services problematic. Typically, a researcher requests a grant for a fixed amount of funding ahead of time. To use a cloud service, one would have to know the rental cost of cloud instances in advance, which can be difficult or impossible. There are mechanisms to estimate it, but they can be such a hassle, you may be better off buying a computer, since it will have a fixed, known cost.

Chapter 6

In closing

I hope your journey through this book has been been fun and productive. Most attempts to learn bioinformatics are quite the opposite in my experience. As mentioned earlier, the goal of this text is to demonstrate an easier and more sustainable approach to bioinformatics analysis. This is, however, still a work in progress. Additional information and improvements will be added here frequently, so be sure to check for free updates to this ebook periodically. If you are using an E-reader, consult the documentation for your device about downloading free updates. Those using a web-based application should see the latest edition automatically.

Best of luck with your future analyses!

Chapter 7

References

[ATAC] Jason D Buenrostro, Beijing Wu, Howard Y Chang, and William J Greenleaf, *ATAC-seq: A Method for Assaying Chromatin Accessibility Genome-Wide*, Copyright © 2015, DOI https://doi.org/10.1002/-0471142727.mb2129s109 , Current Protocols in Molecular Biology.

[Auer] Paul Auer, *Statistical Design and Analysis of RNA Sequencing Data*, Copyright © 2010, DOI https://doi.org/-10.1534/genetics.110.114983 , Genetics.

[Baruzzo] Giacomo Baruzzo, *Simulation-based comprehensive benchmarking of RNA-seq aligners*, Copyright © 2017, DOI https://doi.org/10.1038/ , Nature Methods.

[Donato] Luigi Donato, *New evaluation methods of read mapping by 17 aligners on simulated and empirical NGS data: an updated comparison of DNA- and RNA-Seq data from Illumina and Ion Torrent technologies*, Copyright © 2021, DOI https://doi.org/10.1007/s00521-021-06188-z , Nature Computing and Applications.

[Evans] Ciaran Evans, *Selecting between-sample RNA-Seq normalization methods from the perspective of their assumptions*, Copyright © 2018, DOI https://doi.org/10.1093/bib/bbx008 , Briefings in Bioinformatics.

[Gasperskaja] Evelina Gasperskaja, *The most common technologies and tools for functional genome analysis*, Copyright © 2017, DOI https://doi.org/10.6001/actamedica.v24i1.3457 , Acta medica Lituanica.

[Hisat2] Daehwan Kim, *Graph-based genome alignment and genotyping with HISAT2 and HISAT-genotype*, Copyright © 2019, DOI doi.org/10.1038/s41587-019-0201-4 , Nature Biotechnology.

[Kallisto] Nicolas Bray, *Near-optimal probabilistic RNA-seq quantification*, Copyright © 2016, DOI https://doi.org/-10.1038/nbt.3519 , Nature Biotechnology.

[Krizanovic] Kresimir Krizanovic, *Evaluation of tools for long read RNA-seq splice-aware alignment*, Copyright © 2017, DOI https://doi.org/10.1093/bioinformatics/btx668 , Bioinformatics.

[Li-FalsePos] Yumei Li, Xinzhou Ge, Fanglue Peng, Wei Li, and Jinyi Jessica Li, *Exaggerated false positives by popular differential expression methods when analyzing human population samples*, Copyright © 2022, DOI https://doi.org/10.1186/s13059-022-02648-4 , Genome Biology 23, 79.

[Liao] Liao Yang and Shi Wei, *Read trimming is not required for mapping and quantification of RNA-seq reads at the gene level*, Copyright © 2020, DOI https://doi.org/10.1093/nargab/lqaa068 , NAR Genomics and Bioinformatics.

[STAR] Alexander Dobin, *STAR: ultrafast universal RNA-seq aligner*, Copyright © 2012, DOI https://doi.org/10.1093/-bioinformatics/bts635 , Bioinformatics.

[Schaarschmidt-align] Stephanie Schaarschmidt, *Evaluation of Seven Different RNA-Seq Alignment Tools Based on Experimental Data from the Model Plant Arabidopsis thaliana*, Copyright © 2020, DOI https://doi.org/10.3390/-ijms21051720 , International Journal of Molecular Sciences.

[Schurch] Nicholas Schurch, *How many biological replicates are needed in an RNA-seq experiment and which differential expression tool should you use?*, Copyright © 2016, DOI https://doi.org/10.1261/rna.053959.115 , RNA.

[Sleuth] Harold Pimentel, *Differential analysis of RNA-seq incorporating quantification uncertainty*, Copyright © 2017, DOI https://doi.org/10.1038/ , Nature.

[Zhao-norm] Shanrong Zhao, *Misuse of RPKM or TPM normalization when comparing across samples and sequencing protocols*, Copyright © 2020, DOI https://doi.org/10.1261/rna.074922.120 , RNA.