Foundational Statistics - Bi 610 - Spring 2020

Clayton M. Small and William A. Cresko

2020-04-07

Contents

Chapter 1

Course Overview

This is the complete set of *course materials* for the *Foundational Statistics Course* at the University of Oregon for the Spring of 2020. It is written in **Markdown** so that it can be easily updated.

In this book you will find nearly all the information you will need to complete the course.

Chapter 2

Introduction to the course

This is the complete set of *course materials* for the *Foundational Statistics Course* at the University of Oregon for the Spring of 2020. It is written in **Markdown** so that it can be easily updated.

In this book you will find nearly all the information you will need to complete the course.

2.1 Instructors

Dr. Clay Small, csmall@uoregon.edu

Dr. Bill Cresko, wcresko@uoregon.edu

2.2 Course Information

Virtual Office Hours: T-R 12 to 1:30 (Zoom)

2.3 Software

- Latest version of R
- Latest version of RStudio

2.4 Inclusion and Accessibility

Please tell us your preferred pronouns and/or name, especially if it differs from the class roster. We take seriously our responsibility to create inclusive learning environments. Please notify us if there are aspects of the instruction or design of this course that result in barriers to your participation! You are also encouraged to contact the Accessible Education Center in 164 Oregon Hall at 541-346-1155 or uoaec@uoregon.edu.

We are committed to making this course an inclusive and respectful learning space. Being respectful includes using preferred pronouns for your classmates. Your classmates come from a diverse set of backgrounds and experiences; please avoid assumptions or stereotypes, and aim for inclusivity. Let us know if there are classroom dynamics that impede your (or someone else's) full engagement.

Because of the COVID-19 pandemic, this course is being delivered entirely remotely. We realized that this situation makes it difficult for some students to interact with the material, for a variety of reasons. We are committed to flexibility during this stressful time and emphasize that we will work with students to overcome difficult barriers as they arise.

Please see this page for more information on campus resources, academic integrity, discrimination, and harassment (and reporting of it).

Chapter 3

Course Schedule

3.1 Weeks 1-2

1. Data organization and management
    - best practices, reproducibility, etc.
2. Basic programming fundamentals for data curation
    - The Unix environment and fundamental commands
    - Formatting and manipulating tabular text files from the terminal
3. Introduction to R and Rstudio
    - Installation/Updates
    - R object types and assignment
4. Practice with R objects
    - vectors, matrices, data frames, etc.
5. Applying core programming fundamentals in R
    - vectorized operations
    - replicate, apply family, ifelse, for loops, etc.

3.2 Week 3

1. Plotting/visualizing data as a means of exploration
    - Different plot types
    - Scale, transformations, etc.
2. Fundamentals of plotting in base R
    - par
    - using palettes, points, sizes, etc. to convey information
    - axes and labels
3. R markdown

3.3 Week 4

1. Population parameters, samples, and sampling distributions
    - Central Limit Theorem and the normal dist.
    - Mean and st. dev.
2. Probability and probability distributions
3. Calculating summary statistics
    - Other common summary statistics (quantiles, etc.)

3.4 Week 5

1. Parameter estimation
    - Simulating data sets with known parameters
    - Revisit probability distributions
2. Uncertainty in estimation
    - Parametric and nonparametric approaches to uncertainty

3.5 Week 6

1. Experimental design
    - lexicon
    - considering sources of variance
    - types of variables (categorical, ordinal, rational)
    - confounding variables
2. Frequentist hypothesis testing
    - error types
    - p-values
    - degrees of freedom
    - statistical power
    - multiple testing problem

3.6 Week 7

1. Comparing means between groups
    - Student's t-test
2. Bootstrapping and randomization to compare means

3.7 Week 8

1. Relationships between quantitative variables
    - correlation and covariance

2. Simple linear regression
    - residuals and least squares
    - fitting linear regression models

3.8 Week 9

1. Analysis of variance
    - Table components and test statistics
2. General linear models in R
    - Model formulae
    - Interpretation of summary output
3. More complex ANOVA frameworks
    - Nested models
    - Factorial models

3.9 Week 10

1. Frequency-based statistical tests
    - Chi-squared tests
    - Contingency tables and tests of independence
2. Brief introduction to generalized linear models (time permitting)
    - logistic regression

Chapter 4

Background material for the course

4.1 Description of the course

This course in an introduction to data management, data visualization, and statistical inference. It is intended for early-stage graduate students with no background in statistics. No prior coursework (undergraduate or graduate) in statistics or programming is assumed. The primary objective of the course is to get students up to speed with respect to organization, manipulation, visualization, and analysis of data, using the R statistical language. The emphasis on application is strong, with the goal of enabling students (after the course) to analyze their own data sets with confidence using reasonable approaches, and, when faced with more difficult analysis problems, to be able to communicate their inference objectives clearly to expert analysts. Students will learn to organize and analyze data sets in the form of RStudio projects, using R Markdown files to reproducibly capture and render code, visualizations, and analyses. In-class exercises will be delivered in the form of pre-formatted R Notebooks, which can be interactively executed by students without having to write all code from scratch.

The course is designed to acquaint students primarily with univariate (single response variable) analysis. Multivariate analysis will be covered in the Advanced Biostatics 2-course series offered during the Fall and Winter terms. Examples and assignments in class will include data sets primarily from the biological sciences, including studies of morphological and molecular traits, behaviors, ecological questions, and clinical studies. For specific statistical topics covered in class, please see the course goals and tentative schedule below.

4.2 Course goals:

- Properly organize and format primary data and metadata files for analysis
- Learn programming fundamentals of the R statistical language, including objects, functions, iteration, and simulation.
- Make publication-quality data visualizations, including scatterplots, boxplots, frequency distributions, mosaic plots, etc.
- Understand Type I and Type II statistical error, including p-values and power analysis.
- Understand ordinary least-squares regression and linear models in general
- Learn the fundamentals of strong experimental design
- Learn to apply general linear models to basic univariate analysis problems, including Analysis of Variance (ANOVA)
- Learn nonparametric approaches to parameter estimate and statistical inference, including resampling (bootstrapping), permutation, and rank-based analysis.
- Understand how to analyze binary response variables and frequency-based (e.g. contingency table) data sets.

4.3 Introduction to R and RStudio

R is a core computational platform for statistical analysis. It was developed a number of years ago to create an open source envirnment for advanced computing in statistics and has since become the standard for statistical analysis in the field, replacing commerical packages like SAS and SPSS for the most part. Learning R is an essential part of becoming a scientist who is able to work at the cutting edge of statistical analysis – or even to perform conventional statistical tests (e.g. a t-test) in a standard way. An important part of R is that it is script-based, which makes it easy to create reproducible analysis pipelines, which is an emerging feature of the open data/open analysis movement in science. This is becoming an important component of publication and sharing of research results, so being able to engage fully with this effort is something that all young scientists should do.

RMarkdown is an extra layer placed on top of R that makes it easy to integrate text explanations of what is going on, native R code/scripts, and R output all in one document. The final result can be put into a variety of forms, including webpages, pdf documents, Word documents, etc. Entire books are now written in RMarkdown and its relatives. It is a great way to make quick webpages, like this document, for instance. It is very easy to use and will be the format that I use to distribute your assignments to you and that you will use to turn in your assignments.

R Projects are a simple way of designating a working directory in which to house files related to a given, well, project. Those files might include primary data and metadata files ready for reading into R, `.R` scripts, Rmarkdown files, and output such as Rmarkdown-rendered .html files or individual plots, for example. The nice thing about organizing your work with R Projects is that you can keep everthing needed to reproduce an analysis in a single directory on your computer. You can open an R Project in RStudio by opening the project's index (`.RProj`) file, which will automatically set your working directory to that of the project and facilitate loading any saved environments, etc.

In Chapter 6 we will begin working in R and RStudio, but you can get them installed now (in that order) on your computer, if you haven't already. Get the most recent *released* R version by following this link: https://www.r-project.org/

We will do our work using Rstudio, which is a powerful and convenient user interface for R, and can be downloaded from here for installation: https://rstudio.com/products/rstudio/

4.3.1 Learning resources

There are tons of resources for learning R and RMarkdown on the internet. Here are just a few, but you will no doubt find your own favorites as you become routine R users.

There is an organized group that is dedicated to training in R called DataCamp (https://www.datacamp.com/). They provide all of the basics for free. They actually have training for most data science platforms. RStudio provides links for training directly related to R and RMarkdown here: https://www.rstudio.com/online-learning/

There are also many, many R training videos on YouTube. Most of them are very well meaning but may not be as in-depth as you want.

You can also go the old "paper" manual route by reading the materials provided by R itself: https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf

In reality, if you want to do almost anything in R, simply type in what you are interested in doing into Google and include "in R" and a whole bunch of links telling you exactly what to do will magically appear. Most of them appear as discussions on websites like StackOverflow and Stats.StackExchange. In that case, the first thing that you see is the question–usually someone doing it just a bit wrong–so you should scroll down to see the right way to do it in the answers. It is really an amazing resource that will speed you along in nearly every form of analysis that you are interested in.

Please do not hesitate to contact us if you have questions or run into obstacles. The point of this class is to learn by doing, but our aim is that the doing should involve reasonable first efforts supplemented with help if needed. Also, many of your classmates have some experience with R, writing code, or statistics in general, so they are an excellent resource as well!

Chapter 5

Organizing and manipulating data files

5.1 Introduction

Many of you will already be familiar with data file organizaiton, editing, and formatting for analysis. If so, much of the following material may be review. If not, some of the following guidelines and tools should prove to be quite useful. In biology, and many other fields, primary data are routinely stored as "flat" text files. The exact formatting depends on the type of data, of course, but often we are working with text files organized into rows and columns. Rows can naturally be defined by lines in a file, and columns can be defined by separators (also called delimiters) such as spaces, tabs, or commas, to name a few commonly used ones. Fortunately there are some very powerful and simple-to-use (with a little practice) tools that can be invoked directly from a computer's command line, or included in written "scripts" that your computer's operating system can interpret upon you running them. These command line tools are now nearly ubiquitous on all personal computer platforms. Computers running a LINUX operating system allow direct access to these tools via the command line, as does the macOS operating system of Apple computers via the Terminal. Computers running Microsoft Windows 10 now also facilitate use of these conventional "UNIX tools" through a Windows Subsystem for Linux.

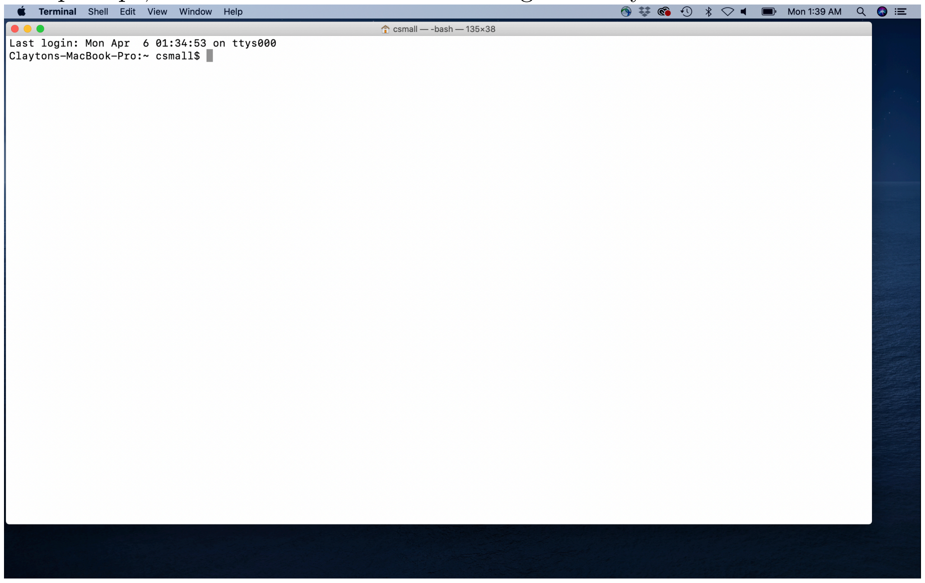
In the following sections, we provide a *very brief* introduction to using some of these tools in order to organize your data files, parse them for information, and perform some basic text manipulations. Mastering these activities is not necessary for this course (in fact, many of the text manipulation tasks can be done in R!), but if you learn to adopt at least some of these skills you will become a better, more organized analyst, and it will help you become comfortable with the command line and programming in general.

5.2 Navigating file systems from the command line

5.2.1 Access to the command line

The first step to using command line tools is to get access to the command line! On Mac and Linux systems you can simply do this by finding and opening the `Terminal` application. On Windows 10 systems, you'll have to install a Linux Bash Shell if you haven't already. To do this you will need to follow the instructions here: https://itsfoss.com/install-bash-on-windows/ When you get to the point of choosing the Linux distribution to install, I recommend Ubuntu.

At this point you should have command line access through a terminal prompt, which should look something like my Mac Terminal below:



```
Last login: Mon Apr  6 01:34:53 on ttys000
Claytons-MacBook-Pro:~ csmall$ 
```

You are now ready to navigate and explore files simply by typing!

5.2.2 Navigating directories and files

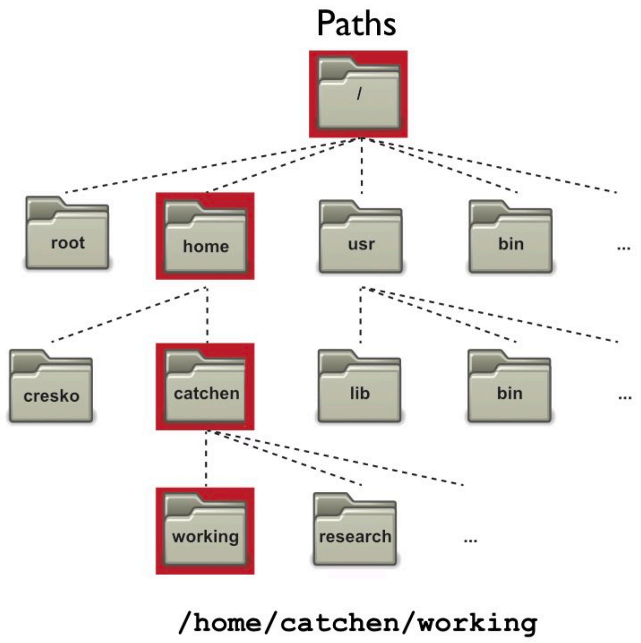
When you are at the command line, just think of your computer as you would if you were navigating using a graphical application (e.g. Mac Finder or Windows Explorer). You are always in a directory in your file system, and you can move to any other directory by typing the approriate command and destination, then hitting Enter.

The first crucial UNIX command to learn is `pwd`. This command stands for "print working directory," and it will literally print the path of the directory you are currently in.

Another important command is `ls`. This lists the files and directories (by default) in your working directory. If you specify a different directory, it will list the files and/or directories there. Most UNIX commands (and indeed command-line programs in general), can be run with options. One way to invoke the and option is to type a "flag" along with the command. In the case of `ls`, we can type `ls -l`, for example, which will print the output line-by-line. We can also add annother flag: `ls -lh` (equivalent to `ls -l -h`), which will print items line-by-line but also make sure the item sizes are "human readable." If you ever have quesitons about how to use UNIX program, including the flags and other options, you can type `man program_name` and a wonderful help manual will appear. To exit and return to the command prompt, just hit "q". These `man` pages are extremely useful and should be your first go-to if you need information for a particular command. Please use these regularly!

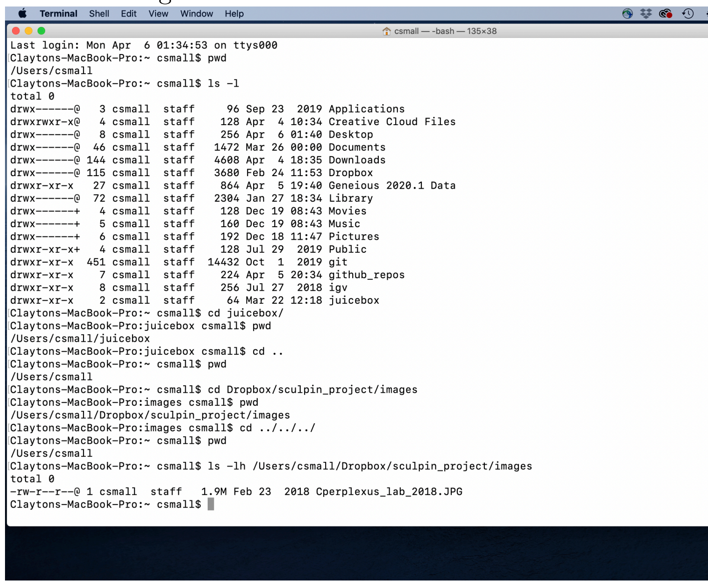
The command `cd` will change your location from the current directory to another directory. Like many other programs (UNIX and otherwise) require you to input directory and file locations, with `cd` you can specify your desired location using either the *absolute* or *relative* path. An absolute path is the full "address" of a directory or file, starting from the root of your file system. An example of an absolute path to a directory in my file system is `/Users/csmall/Dropbox/sculpin_project/images/`. Regardless of where my current working directory is in my file system, I can change to this `images/` directory using `cd` and the full path. I can also use a relative path, which is a sort of "shortcut," to specify the location of a directory or file. Let's say I am in `/Users/csmall/Dropbox/BiostatsFound_S2020/` and I want to get to the `images/` directory above. I could type `cd ../sculpin_project/images`, which uses a relative path to take me "up" one directory (as denoted by `../`) into `Dropbox/` and back "down" into `sculpin_project/images`. In fact, `..` is a special file in every directory that just means "the directory above." The special file `.` is the current directory. And to mention one final useful designation for navigation shortcuts, you can use the `~` to denote your home directory.

The schematic below should help you visualize how to think about file system



navigation from the commmand line:

And for another example, take a look at this series of navigation commands from



```
Last login: Mon Apr  6 01:34:53 on ttys000
Claytons-MacBook-Pro:~ csmall$ pwd
/Users/csmall
Claytons-MacBook-Pro:~ csmall$ ls -l
total 0
drwx------@   3 csmall  staff     96 Sep 23  2019 Applications
drwxrwxr-x@   4 csmall  staff    128 Apr  4 10:34 Creative Cloud Files
drwx------@   8 csmall  staff    256 Apr  6 01:40 Desktop
drwx------@  46 csmall  staff   1472 Mar 26 00:00 Documents
drwx------@ 144 csmall  staff   4608 Apr  4 18:35 Downloads
drwx------@ 115 csmall  staff   3680 Feb 24 11:53 Dropbox
drwxr-xr-x   27 csmall  staff    864 Apr  5 19:40 Geneious 2020.1 Data
drwx------@  72 csmall  staff   2304 Jan 27 18:34 Library
drwx------+   4 csmall  staff    128 Dec 19 08:43 Movies
drwx------+   5 csmall  staff    160 Dec 19 08:43 Music
drwx------+   6 csmall  staff    192 Dec 18 11:47 Pictures
drwxr-xr-x+   4 csmall  staff    128 Jul 29  2019 Public
drwxr-xr-x  451 csmall  staff  14432 Oct  1  2019 git
drwxr-xr-x    7 csmall  staff    224 Apr  5 20:34 github_repos
drwxr-xr-x    8 csmall  staff    256 Jul 27  2018 igv
drwxr-xr-x    2 csmall  staff     64 Mar 22 12:18 juicebox
Claytons-MacBook-Pro:~ csmall$ cd juicebox/
Claytons-MacBook-Pro:juicebox csmall$ pwd
/Users/csmall/juicebox
Claytons-MacBook-Pro:juicebox csmall$ cd ..
Claytons-MacBook-Pro:~ csmall$ pwd
/Users/csmall
Claytons-MacBook-Pro:~ csmall$ cd Dropbox/sculpin_project/images
Claytons-MacBook-Pro:images csmall$ pwd
/Users/csmall/Dropbox/sculpin_project/images
Claytons-MacBook-Pro:images csmall$ cd ../../../
Claytons-MacBook-Pro:~ csmall$ pwd
/Users/csmall
Claytons-MacBook-Pro:~ csmall$ ls -lh /Users/csmall/Dropbox/sculpin_project/images
total 0
-rw-r--r--@ 1 csmall  staff   1.9M Feb 23  2018 Cperplexus_lab_2018.JPG
Claytons-MacBook-Pro:~ csmall$ 
```

my terminal and see if you can follow along:

If you want to create a new directory, you can use the `mkdir` command, including the desired name of the new directory. By default this will create the directory in your current working directory, but you can use absolute or relative paths to instead write the directory somewhere else. If you want to delete an empty directory, `rmdir` is the appropriate command.

Now let's briefly cover some UNIX commands that are useful for managing files. Some of these apply to directories as well, which I will point out as we go. The command `touch` can be used to create a new, empty file, which you can add to using a plain text editor. Examples of popular plain text editors with advanced user interfaces are BBEdit and Atom. You can also use command line text editors, such as `nano`, `emacs`, and `vim`. Most UNIX/LINUX systems have `nano` installed by default. To copy or change the name and/or location of a file (or directory), use `cp` and `mv` commands, respectively. Note that by using absolute or relative paths, you can specify where you want the file or directory to end up. Be especially careful with these, however, because you will overwrite any existing file or directory if you specify the same name and location. Another command you should be extremely cautious with is `rm`, which removes (permanently deletes) a file. `rm -r` can be used to delete a non-empty directory AND all of its contents.

In many cases you will want to look at files, or parts of them at least, from the command line. `cat` will print the entire contents of a file, but can also be used to combine ("concatenate") multiple files in a line-wise manner. `less` and `more` will display specific lines of a file (starting with the first ones), with single- or multi-line "scrolling," respectively, activated using the return or down-arrow keys. To leave the display, you need to hit the "q" key. `head` and `tail` will display the first or last, respectively, *n* lines of the file, where *n* is provided as a flag (e.g. `head -200 file.tsv`). The "word count" command `wc` can quantify elements of a text file in various ways, but one common application is `wc -l`, which counts the number of lines in a file.

An aside: If you are working from the command line and want to terminate a process (say you accidentally start a task that will take way too long), press Ctrl-C.

5.2.2.1 A quick review of important UNIX commands for navigation and viewing

`pwd` - prints working directory

`ls` - lists contents of a directory

`cd` - changes the working directory

`mkdir` - creates a new directory

`rmdir` - deletes an empty directory

`touch` - creates an empty file

`cp` - copies a file or directory

`mv` - changes the name of a file or directory `rm` - deletes a file, or a directory and everyting inside with `-r`

`cat` - prints the entire file to the terminal, or concatenates and prints multiple files

`less` - displays the first lines of a file, with scrolling line-by-line

`head` - prints the first 10 lines (default) of a file

`tail` - prints the last 10 lines (default) of a file

`wc -l` - prints the number of lines in a file

5.2.3 Useful UNIX commands for file manipulation

In many cases you will want to search for specific characters or combinations of characters, and do various things with that information. Maybe you want to isolate the lines of a file that contain the query, or perhaps you want to count how many lines contain the query. The tool `grep` is extremely useful in this regard. We don't have time for a comprehensive dive into the utilities of `grep`, but a few common applications are worth mentioning. Character patterns we search for using `grep` may or may not involve special characters that are not interpretted literally. Here we will discuss just a few common cases of `grep` searches and the special characters involved. Some examples of these special characters include `^` (beginning of a line), `$` (end of a line), `.` (any single character except a newline), `*` (zero or more instances of the preceeding character), and `\s` (any white space). The standard syntax for `grep` from the command line is `grep "expression" filename`. So, if you wanted to return all of the lines in the data file `zfish_data.tsv` (assuming it is in the current directory) that begin with "embryo_10", you could try `grep "^embryo_10" zfish_data.tsv`. This search would also (unintentionally) find lines beginning with "embryo_100" or "embryo_101", etc., if they exist. So, you have to be careful, and learning the rules just takes practice. In this case `grep "^embryo_10\s" zfish_data.tsv` would acheive the desired result, assuming that there is a whitespace delimiter between fields ("columns") in the data file. Useful flags for `grep` include `-c` (which counts the number of lines containing the query), `-v` (which returns the lines that *do not* contain the query), and `-n` (which prints the line number for each line containing the query). I encourage you to look at many different `grep` use cases online as your demand for complex searches grows.

The program `sed` has reasonably complex applciations, but is commonly used as a sort of "search and replace" tool. The syntax for `sed` use is similar to `grep`, except that the query and replacement expressions are organized (with other information) using slashes. For "search and replace" functionality, that sytax looks like this: `sed 's/query/replacement/flag' filename`. One common option for the "flag" component is "g", meaning "global", which replaces all instances. If no flag designation is made only the first instance in the file is replaced. Building on our toy example from above, `sed 's/^embryo_/^larva_/g' zfish_data.tsv` would perform a global replacement and print the output to the terminal. To change the contents in the original file on the fly, including `sed -i` would do the trick, but is riskier than redirecting the output to a new file.

`cut` is quite straightforward, and can be used to isolate individual fields (think of them like "columns") from a text file, provided the fields are consistently separated by a delimeter on each line. So, if I had a comma-separated file and I just wanted the first two columns I could type `cut -f1,2 -d"\t" filename`. Note that if you don't specify a delimter using the `-d` flag, then it is assumed to be tab-delimited. If you want to bring together fields in separate files, `join` can be used to accomplish this. The two files should have equivalent rows, however, for this action to work properly.

If you want to sort text files alphanumerically, in a field-wise fashion, `sort` is quite useful. If a file contains a single field, minimal specification is required, aside from tuning numerical sorting. For example, if you want to sort numerically, use the `-n` flag, and if you want to sort from largest to smallest, add the `-r` flag. If you want to sort a multi-field file based on just one field, you can use the "key" flag. For instance, if you have a tab-delimited file and want to sort by the second field in reverse numerical order, `sort -k2,2 -nr filename.tsv` would give you the desired result. Finally, if you want to eliminate lines with the same value for a given field, you can use the `-u` "unique" flag.

The UNIX program `awk` is an extremely powerful tool, and can itself be used essentially as a mini programming language. We will not get into the myriad uses of `awk` here, but the reference at the bottom of the chapter is a great resource if you want to learn more. `awk` is extremely efficient at parsing and capturing text files in a column-wise manner, with the ability to also evaluate logical statements applied to rows. The structure of `awk` commands is more complex than that of other UNIX programs we have discussed, but it is still very intuitive. One unique feature is that `awk` contains its own internal functions, which are typed inside curly braces. The "print" function can be used to extract fields, much like `cut`. For instance, `awk -F: '{print $1,$6}' filename.tsv` would print the first and sixth field from `filename.tsv`, assuming a ":" delimiter. With `awk`, fields are specified using the `$` character. If you want also to select only specific rows from a set of columns (like those with a certain value), you can incorporate logical operators. In the above example if we had wanted fields 1 and 6, but only those rows with a value of at least 610 in field 2, we could type the following `awk -F: '$4 >= 610 {print $1,$6}' filename.tsv`. Again, this is just scratching the surface with `awk`, which boasts a great deal of potential for your text file manipulation needs.

5.2.3.1 A quick review of key UNIX commands for text file searching and manipulation

`grep` - searches a file for characters and character combinations

`sed` - stream edits characters and character combinations `cut` - isolates specific fields (“columns”) from a file using a delimiter

`join` - combines fields (“columns”) from multiple files with equivalent rows

`sort` - orders the rows in a file based on one or more fields

`awk` - flexibly parses, evaluates, and selectively prints row- and column-wise

5.2.4 A quick word on pipes and carrots

One very convenient feature of UNIX commands is that you can control the flow of input and output from one command to another using the `|` (“pipe”) character. For instance, I may want to search an entire file for rows that begin with “fish-1”, and then replace the “-” with ”_“. To do this I could do something like `cat file.tsv | grep "^fish-1" | sed 's/fish-1/fish_1/g'` This, of course, would print the output to the terminal, but I could actually capture that output into a file using the `>` charcter. `cat filename | grep "^fish-1" | sed 's/fish-1/fish_1/g' > ./newfile.tsv` would write this new file to my current working directory. Furthermore, if you want to append lines of text to an existing file, the”double sideways right-pointing carrot” character `>>` can be used.

The above lessons on UNIX commands for file manipulation truly just scratch the surface of what can be accomplished at the command line and in “shell scripts.” You certainly will have further questions and be hungry for more, but we simply don’t have time during this course. But to work on your UNIX skills for now, check out `Ex1_Unix_Intro.html` (on Canvas). We need to move on to R now, but at the bottom of this chapter are some UNIX command resources I have found to be especially useful.

5.3 Data file and data file entry dos and don’ts

Do store a copy of your data in a nonproprietary format, such as plain ASCII text (aka a flat file). This is especially important if you are using tools (like UNIX commands) to parse and manipulate the files. Formats like Microsoft Excel are not acceptable as input for many analysis tools, and not everyone has access to proprietary software.

Do leave an un-edited copy of an original data file, even when main analyes require an edited version.

Do use descriptive names for your data files and variables, and use them consistently!

Do maintain effective metadata about the data.

Do add new observations to a data file as rows.

Do add new variables to a data file as columns.

Don't include multiple data types in the same column.

Don't use non-alphanumeric characters (other than the underscore) in file or directory names.

Don't use spaces, tabs, commas, colons, semicolons, or other chacters commonly used as field (column) delimiters in names of individual data entries. For example, don't use something like `March 8` as a value for date in a data set.

Don't copy and paste data directly from rich-text-formatted files (like Microsoft Word) into primary data files.

5.4 Exercises associated with this chapter:

- Exercise 1 (file: `Ex1_Unix_Intro.html`)

5.5 Additional learning resources

- http://mally.stanford.edu/~sr/computing/basic-unix.html - A nice "cheat sheet"
- http://korflab.ucdavis.edu/Unix_and_Perl/ - Outstanding tutorial by Keith Bradnam and Ian Korf
- https://www.datacamp.com/courses/introduction-to-shell-for-data-science - DataCamp tutorial
- https://www.gnu.org/software/gawk/manual/gawk.html - A comprehensive guide to `awk`