Analysis of Next Generation Sequencing Data
Introduction to R Markdown and Unix

Luce Skrabanek

15 January, 2019

1 Introduction to R Markdown

1.1 What is R?

R is a free software environment for statistical computing and graphics (`www.r-project.org`). It can effectively analyze large-scale datasets, such as those resulting from high-throughput sequencing experiments. It promotes automated and reproducible analyses of scientific data, creates a wide spectrum of publication quality figures, and has an extensive library of add-on packages to facilitate many complex statistical analyses. Because it is free and ubiquitously available (it runs on Windows, Mac, and Linux computers), your investment in learning R will pay dividends for years to come.

1.2 What is RStudio?

While R is very powerful, it is essentially a command line program and is thus not the friendliest thing to use. Especially when learning R, a friendlier environment is helpful, and RStudio provides this, giving you things you expect in a modern interface like integrated file editing, syntax highlighting, code completion, smart indentation, tools to manage plots, browse files and directories, visualize object structures, etc.

From your computer, choose the RStudio application. This will start R under the hood for you.

2 Staying organized

A good practice is to create a new RStudio project for every project. This is RStudio's way of supporting and streamlining the common practice of keeping all the input data, R scripts, and other files associated with that analysis together. If you tell R about this directory, it will, by default, load and save files from it. We call this the working directory. You can browse files and directories from the Files tab of the lower right panel. This will create a .Rproj file in your project directory. Now, whenever you want to work on that analysis again, opening that .Rproj file will bring you back to exactly where you left off (the same working directory, the same files open, the same command history).

2.1 R Markdown

This is a way of combining not just R code, but also notes and comments about the code and project. It can be thought of as akin to a lab notebook, which you can use to capture scientific ideas and the associated analysis results and communicate these with colleagues.

To start a new R Markdown document, choose *File* ⇒ *New File* ⇒ *R Markdown*, or select R Markdown from the green cross icon. You will be asked to fill in some basic metadata (name of document and author) and what you want the default output format to be.

There are three parts to an R Markdown document:

1. a YAML header, surrounded by triple dashes, which is automatically generated for you when you open a new document

   ```
   ---
   title: "Some title"
   author: "Luce"
   output: html_document
   ---
   ```

2. the chunks of R code that will be run, surrounded by triple backticks (```)

   ```
   ``` {r optional_name, optional_options}
   R code here
   ```
   ```

3. text, with Markdown formatting.

To produce a complete report including code and text, click the Knit button. If you haven't already saved the file, it will ask you to do so now. Try this now, with the default content given to you on creation.

By default, an HTML file will be produced, with a preview either in the Preview panel, or externally (which preference can be set from the Options menu (the cog icon)). The HTML file can be sent to colleagues or your PI, and viewed in a browser. Other output options include PDF and Word, although you will need to install separate packages for that functionality. We will talk more about external packages later.

The sample R Markdown document contains chunks of R code. We will understand this later, but for now, realize that each R code chunk can be run separately. The output from each chunk can

include both the code and the results (use `echo=TRUE` as a chunk option) or just the results (use `echo=FALSE`). The output, including figures, may be shown inline (in the .Rmd file), but many people find this distracting, and prefer to see the output in the Console and Plot panels, which can be set from the Options menu (select Chunk Output in Console).

It is also possible to include mini-code chunks inline, surrounded by single backticks, and prefixed by the letter `r`. These will get evaluated, and get filled in, whenever the document is knitted.

What we care about for today is the text. The formatting of the text is accomplished with standard Markdown formatting. Examples include:

- Different level headings are prefixed by varying numbers of `#`.
- Italic or bold text is surrounded by one and two `*`, respectively,
- Bulleted list items are prefixed by `*`,
- Numbered list items are prefixed by numbers.

There are a number of guides to help you construct your R Markdown documents directly from within RStudio:

- an external R Markdown cheatsheet PDF, which lists all the most commonly used commands and syntax, accessed from *Help* ⇒ *Cheatsheets* ⇒ *R Markdown Cheat Sheet*,
- an external markdown reference guide PDF, accessed from *Help* ⇒ *Cheatsheets* ⇒ *R Markdown Reference Guide*, also found inline at *Help* ⇒ *Markdown Quick Reference*.

3 Introduction to Unix

UNIX is the dominant operating system for high-performance, scientific computing. UNIX was originally developed at AT&T Bell Labs in 1969. There are now many flavors of UNIX. Some common aliases are: IRIX, Solaris, AIX, HP-UX, BSD, Linux (Red Hat, SUSE, Debian, Mandrake, etc), Mac OS X. There are a couple of different UNIX shells. They all fall into two families, the Bourne shell family and the C shell family. Whenever possible, use `bash` or `tcsh`. These are the latest and greatest members of the two families; others are lesser forms of these. The `tcsh` shell has historically been preferred by scientists. The `bash` shell is preferred by computer geeks. Traditionally, scripts are written in `sh` or `bash`. We will be using bash in this class.

3.1 Looking at files (`ls, cat, more, less, head, tail`)

In Unix, (almost) everything is a file. Use the `ls` command to see what files you have.

```
ls
```

Most common UNIX commands are usually a just few letters long. This savz kystrks. Although this makes UNIX appear cryptic and a little harder to learn, you will find you'll get to know these commands quickly. In the long run, you will be much more efficient.

The `cat` command will show you the contents of a file.

```
cat hello_slurm.bash
```

Note that case matters in UNIX.

When you're typing commands, you can use the following:

1. The left and right arrow keys move the cursor left and right (surprising, eh?)
2. The up and down arrow keys scroll forward and backward in your history of commands. This is useful if you need to type a command that is similar to one you previously ran.
3. [TAB] autocompletes a (partial) unambiguous filename.
4. CTRL-A moves to the beginning of the line.
5. CTRL-E moves to the end of the line.
6. CTRL-D deletes the character that the cursor is on.
7. [Delete] works as expected.
8. CTRL-C abandons the whole affair and lets you try again.

You can look at other people's files.

```
cat /home/luce/angsd/quotation
cat ~luce/angsd/quotation
```

But this is not useful when viewing very big files.

```
cat ~luce/angsd/demo.fastq
```

For large files, we can use `more`. With `more`, to advance a line at a time, press `[Return]`. To advance a screenful, press the space bar. To quit the pager, press `q`. Some UNIX systems have a

better pager called `less` (because "less is more") which lets you scroll up too, but it is not on all UNIX systems.

```
more ~luce/angsd/demo.fastq
less ~luce/angsd/demo.fastq
```

We can also use the commands `head` and `tail` to show just a few lines at the beginning or end of the file.

```
head -n 10 ~luce/angsd/demo.fastq
tail -n 25 ~luce/angsd/demo.fastq
```

3.2 Directories (`pwd`, `cd`, relative and absolute pathnames)

Directories are hierarchical, similar to Mac or PC files and folders. In UNIX, the `/` separates the folder names from the file name. There are no drive letters of volume names. The root directory is just `/` and the current directory is `.` and the parent directory is `..` (two periods).

To find out what your current directory is, use the `pwd` command.

To change your current working directory, use the `cd` command. `cd` by itself is shorthand for going directly to your home directory.

```
# by itself, cd is shorthand for going directly to your home directory.
cd
# go to my home directory
cd /home/luce
# from there, go to my ANGSD directory
cd angsd
# shorthand to go back to your home directory
cd ~
```

3.3 Manipulating Files and Directories (`cp`, `mkdir`, `mv`)

We can now see what files we have and look at their contents. Next we will learn how to move files around in the directory structure. Moving files around is equivalent to dragging and dropping files on your desktop computer. However, since we can't use a mouse with UNIX, we need some new commands.

To copy a file, use `cp`.

```
# To copy the demo.fastq file from my account
cp ~luce/angsd/demo.fastq .
```

Make sure you're in your home directory before you do this. The period indicates that the file will be copied with the same name to the current directory.

Let's create a directory to put our newly acquired file into. We do this with the `mkdir` and `mv` commands.


```
mkdir data
mv demo.fastq data
```

We can also rename files using the `mv` command.

```
mv demo.fastq human_expt.fastq
```

What do you need to do before you can execute this command? [HINT: what's your pwd; where's your file?] Can you think of a way in which you could have renamed the file without `cd`'ing into the directory?

```
mv data/demo.fastq data/human_expt.fastq
mv data/demo.fastq human_expt.fastq
```

Note that the second command would have renamed the file and moved it up into the current directory.

```
# To copy the demo.fastq file from my account
cp ~luce/angsd/demo.fastq data/new_demo.fastq
```

You can also move multiple files at the same time, but note that when you are moving files this way, the destination must be a directory, and you can't rename files while you're moving them.

You can also move and rename directories with the same commands.

3.4 Deleting Files and Permissions (`rm`, `rmdir`, `chmod`)

You can remove files using the `rm` command. There is NO UNDO. If you remove a file, you're not getting it back.

To delete a directory, use the `rmdir` command. Note that the directory must first be empty.

We saw before that UNIX is a multi-user operating system. There needs to be a mechanism in place to ensure that you don't corrupt (or delete) other people's files (even if you can see and read them), and that other people don't corrupt (or delete) yours. To do this, each file and directory has a series of permissions associated with it. This tells the system which people can and can't read, write and execute those files.

We've already learnt about the `ls` command. The `ls` command has a -l option which shows you a lot of information about the files and directories, including the permissions associated with them. Let's try to understand the output from `ls -l`:

```
-rw-r--r--. 1 luce abc 5753003 Jan 11 16:48 demo.fastq
-rw-r--r--. 1 luce abc  111671 Jan 14 14:33 gene_counts_long.txt
-rw-r--r--. 1 luce abc    2019 Jan 14 14:18 markdown_quick_reference
-rw-r--r--. 1 luce abc    5807 Jan 14 14:47 new_sample.counts
-rw-r--r--. 1 luce abc     132 Jan 11 16:39 quotation
-rw-r--r--. 1 luce abc      81 Jan 11 16:39 Quotation
drwxr-xr-x. 2 luce abc    4096 Jan 11 17:28 slurm/
```


The last column is the filename or directory name, which is what appears when you do an ordinary `ls` command. Preceding that, is the date and time at which that file or directory was last modified. Continuing to work backwards, we see the size of the file or directory in bytes. The next column is the group owner (we'll say more about this in a moment). Then we have the username of the owner of the file. If you own the file, this will be your username.

At the beginning of the line are a series of 10 letters or hyphens which show the filetype and permissions. The first character gives the file type: directories are shown with a `d` and regular files with a hyphen. The remaining letters are `r`, `w`, and `x` and represent 'read', 'write' and 'execute' permissions, respectively. There are three sets of these letters:

1. The first set shows the permissions that the owner himself has for this file.
2. The second set indicates the permissions for the group of users that the user belongs to.
3. The third set shows the permissions for all other users who have accounts on the system.

You can specify who can read, write or execute your files by using the `chmod` command. The user is referred to by `u`, the group by `g` and all other users by `o`. You can add, or take away, permissions using `+` and `-` so long as you are the owner of the file.

```
# Make your slurm batch file executable by you only
chmod go-x hello_slurm.bash
# Make the same file readable and writable by everybody.
chmod go+rw hello_slurm.bash
```

Note that the system administrator can read anybody's file so this is not a solution to true privacy.

3.5 Piping commands

The `wc` command (word count) counts lines, words, and characters in a file.

```
cat ~luce/angsd/quotation

THERE IS NOTHING NOBLE IN BEING SUPERIOR TO SOME OTHER
MAN. TRUE NOBILITY IS BEING SUPERIOR TO YOUR FORMER SELF.
  -- HINDU PROVERB

wc quotation
3 23  132 quotation
```

This reports 3 lines, 23 "words", and 132 characters in this proverb. Count the words yourself. Did you get 23? Why do you think the `wc` command reports 23 words? What is the definition of a word that the `wc` command uses?

To learn more about a command, use the `man` command.

```
man wc
```

This tells you everything you wanted to know about a command, and a lot more besides. The 'man pages' can be a bit dry, but are usually very precise.

When the output of the man command is longer than a screenful, it is "piped" into the `less` command, so you can view it one page at a time. This idea of "piping" the output of one command into the input of another is a key concept in getting the most out of UNIX.

The `man` command pipes its output to the `less` command silently, but usually we need to specify this behavior explicitly. This is done with the `|` operator, called the "pipe" character.

We can use pipes to figure out how many words there are in the first 25 lines of the markdown quick reference guide.

```
# Step 1: What are the first 25 lines of the reference guide?
head -n 25 ~luce/angsd/markdown_quick_reference
# Step 2: How many words are in the result of the previous command?
wc -w
# Steps 1 & 2 together!!!
head -n 25 ~luce/angsd/markdown_quick_reference | wc -w
```

The more commands you know, the better off you are. Your capabilities grow exponentially as does your ability to combine the commands you know.

3.6 Manipulating Data (`cut`, `paste`, `join`, `sort`, `uniq`, `wget`)

The `cut` command is used to extract selected columns or fields from a file.

When processing by field, lines that do not contain any field delimiters will be passed though to `stdout` untouched. This behavior can be overridden by using the `-s` option, which suppresses those lines.

```
# Example: Extract the Geneid and gene counts from gene_counts_long.txt
cat gene_counts_long.txt | cut -f 1,7-12 > gene_counts.txt
cat gene_counts_long.txt | cut --complement -f 2-6 | head
```

The `paste` command merges sequentially corresponding lines from different files. Note that this should not be used to merge files on a common field. The `join` command can be used for this purpose.

```
# Example: print the gene counts from Sample_7 beside the counts from
    Samples 1-6
paste gene_counts.txt new_sample.counts | head
join gene_counts.txt new_sample.counts | head
```

The `sort` command sorts files. Some of the commonly used options for `sort` include:

1. You can specify which field to sort by using the `-k` option.
2. `sort` assumes that fields are separated by whitespace. You can change the field delimiter with the `-t` option.
3. To sort in reverse order, use the `-r` option.
4. To sort in numerical order, use the `-n` option.


```
# Example: sort all the files in your directory by size:
ls -s | sort -n

# Example: sort the gene counts by the number of reads in Sample_1,
   highest to lowest
sort -rn -k 2 gene_counts.txt | head
```

Another interesting command to know about is the `uniq` command. It helps you find unique lines of fields. You might use the `uniq` command, for example, to list out the different kinds of keywords that a PDB file can begin with, or to list the types of amino acids that appear in such a protein structure file.

For your homework, you will have to download data from the web and manipulate it. To download web content, use the `wget` command.

```
# Example: download the p53 protein from Unigene
wget https://www.uniprot.org/uniprot/P04637.fasta
```

