Automating a Variant Calling Workflow

Overview

Teaching: 30 min
Exercises: 15 min
Questions

- How can I make my workflow more efficient and less error-prone?

Objectives

- Write a shell script with multiple variables.
- Incorporate a `for` loop into a shell script.

What is a shell script?

You wrote a simple shell script in a previous lesson (http://www.datacarpentry.org/shell-genomics/05-writing-scripts/) that we used to extract bad reads from our FASTQ files and put them into a new file.

Here's the script you wrote:

Bash

```
grep -B1 -A2 NNNNNNNNNN *.fastq > scripted_bad_reads.txt

echo "Script finished!"
```

That script was only two lines long, but shell scripts can be much more complicated than that and can be used to perform a large number of operations on one or many files. This saves you the effort of having to type each of those commands over for each of your data files and makes your work less error-prone and more reproducible. For example, the variant calling workflow we just carried out had about eight steps where we had to type a command into our terminal. Most of these commands were pretty long. If we wanted to do this for all six of our data files, that would be forty-eight steps. If we had 50 samples (a more realistic number), it would be 400 steps! You can see why we want to automate this.

We've also used `for` loops in previous lessons to iterate one or two commands over multiple input files. In these `for` loops, the filename was defined as a variable in the `for` statement, which enabled you to run the loop on multiple files. We will be using variable assignments like this in our new shell scripts.

Here's the `for` loop you wrote for unzipping `.zip` files:

Bash

```
$ for filename in *.zip
> do
> unzip $filename
> done
```

And here's the one you wrote for running Trimmomatic on all of our `.fastq` sample files:

Bash

```
$ for infile in *_1.fastq.gz
> do
>   base=$(basename ${infile} _1.fastq.gz)
>   trimmomatic PE ${infile} ${base}_2.fastq.gz \
>                ${base}_1.trim.fastq.gz ${base}_1un.trim.fastq.gz \
>                ${base}_2.trim.fastq.gz ${base}_2un.trim.fastq.gz \
>                SLIDINGWINDOW:4:20 MINLEN:25 ILLUMINACLIP:NexteraPE-PE.fa:2:40:15
> done
```

Notice that in this `for` loop, we used two variables, `infile`, which was defined in the `for` statement, and `base`, which was created from the filename during each iteration of the loop.

Creating Variables

Within the Bash shell you can create variables at any time (as we did above, and during the 'for' loop lesson). Assign any name and the value using the assignment operator: '='. You can check the current definition of your variable by typing into your script: echo $variable_name.

In this lesson, we'll use two shell scripts to automate the variant calling analysis: one for FastQC analysis (including creating our summary file), and a second for the remaining variant calling. To write a script to run our FastQC analysis, we'll take each of the commands we entered to run FastQC and process the output files and put them into a single file with a `.sh` extension. The `.sh` is not essential, but serves as a reminder to ourselves and to the computer that this is a shell script.

Analyzing Quality with FastQC

We will use the command `touch` to create a new file where we will write our shell script. We will create this script in a new directory called `scripts/`. Previously, we used `nano` to create and open a new file. The command `touch` allows us to create a new file without opening that file.

Bash

```
$ mkdir -p ~/dc_workshop/scripts
$ cd ~/dc_workshop/scripts
$ touch read_qc.sh
$ ls
```

Output

```
read_qc.sh
```

We now have an empty file called `read_qc.sh` in our `scripts/` directory. We will now open this file in `nano` and start building our script.

Bash

```
$ nano read_qc.sh
```

Enter the following pieces of code into your shell script (not into your terminal prompt).

Our first line will ensure that our script will exit if an error occurs, and is a good idea to include at the beginning of your scripts. The second line will move us into the `untrimmed_fastq/` directory when we run our script.

Output

```
set -e
cd ~/dc_workshop/data/untrimmed_fastq/
```

These next two lines will give us a status message to tell us that we are currently running FastQC, then will run FastQC on all of the files in our current directory with a `.fastq` extension.

Output

```
echo "Running FastQC ..."
fastqc *.fastq*
```

Our next line will create a new directory to hold our FastQC output files. Here we are using the `-p` option for `mkdir` again. It is a good idea to use this option in your shell scripts to avoid running into errors if you don't have the directory structure you think you do.

Output

```
mkdir -p ~/dc_workshop/results/fastqc_untrimmed_reads
```

Our next three lines first give us a status message to tell us we are saving the results from FastQC, then moves all of the files with a `.zip` or a `.html` extension to the directory we just created for storing our FastQC results.

Output

```
echo "Saving FastQC results..."
mv *.zip ~/dc_workshop/results/fastqc_untrimmed_reads/
mv *.html ~/dc_workshop/results/fastqc_untrimmed_reads/
```

The next line moves us to the results directory where we've stored our output.

Output

```
cd ~/dc_workshop/results/fastqc_untrimmed_reads/
```

The next five lines should look very familiar. First we give ourselves a status message to tell us that we're unzipping our ZIP files. Then we run our for loop to unzip all of the `.zip` files in this directory.

Output

```
echo "Unzipping..."
for filename in *.zip
    do
    unzip $filename
    done
```

Next we concatenate all of our summary files into a single output file, with a status message to remind ourselves that this is what we're doing.

Output

```
echo "Saving summary..."
cat */summary.txt > ~/dc_workshop/docs/fastqc_summaries.txt
```

Using `echo` statements

We've used `echo` statements to add progress statements to our script. Our script will print these statements as it is running and therefore we will be able to see how far our script has progressed.

Your full shell script should now look like this:

Output

```
set -e
cd ~/dc_workshop/data/untrimmed_fastq/

echo "Running FastQC ..."
fastqc *.fastq*

mkdir -p ~/dc_workshop/results/fastqc_untrimmed_reads

echo "Saving FastQC results..."
mv *.zip ~/dc_workshop/results/fastqc_untrimmed_reads/
mv *.html ~/dc_workshop/results/fastqc_untrimmed_reads/

cd ~/dc_workshop/results/fastqc_untrimmed_reads/

echo "Unzipping..."
for filename in *.zip
    do
    unzip $filename
    done

echo "Saving summary..."
cat */summary.txt > ~/dc_workshop/docs/fastqc_summaries.txt
```

Save your file and exit `nano`. We can now run our script:

Bash

```
$ bash read_qc.sh
```

Output

```
Running FastQC ...
Started analysis of SRR2584866.fastq
Approx 5% complete for SRR2584866.fastq
Approx 10% complete for SRR2584866.fastq
Approx 15% complete for SRR2584866.fastq
Approx 20% complete for SRR2584866.fastq
Approx 25% complete for SRR2584866.fastq
.
.
.
```

For each of your sample files, FastQC will ask if you want to replace the existing version with a new version. This is because we have already run FastQC on this samples files and generated all of the outputs. We are now doing this again using our scripts. Go ahead and select `A` each time this message appears. It will appear once per sample file (six times total).

Output

```
replace SRR2584866_fastqc/Icons/fastqc_icon.png? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

Automating the Rest of our Variant Calling Workflow

We can extend these principles to the entire variant calling workflow. To do this, we will take all of the individual commands that we wrote before, put them into a single file, add variables so that the script knows to iterate through our input files and write to the appropriate output files. This is very similar to what we did with our `read_qc.sh` script, but will be a bit more complex.

Download the script from here (https://raw.githubusercontent.com/datacarpentry/wrangling-genomics/gh-pages/files/run_variant_calling.sh). Download to `~/dc_workshop/scripts`.

Bash

```
curl -O https://raw.githubusercontent.com/datacarpentry/wrangling-genomics/gh-pages/files/run_v
ariant_calling.sh
```

Our variant calling workflow has the following steps:

1. Index the reference genome for use by bwa and samtools.
2. Align reads to reference genome.
3. Convert the format of the alignment to sorted BAM, with some intermediate steps.
4. Calculate the read coverage of positions in the genome.
5. Detect the single nucleotide polymorphisms (SNPs).
6. Filter and report the SNP variants in VCF (variant calling format).

Let's go through this script together:

Bash

```
$ cd ~/dc_workshop/scripts
$ less run_variant_calling.sh
```

The script should look like this:

Output

```
set -e
cd ~/dc_workshop/results

genome=~/dc_workshop/data/ref_genome/ecoli_rel606.fasta

bwa index $genome

mkdir -p sam bam bcf vcf

for fq1 in ~/dc_workshop/data/trimmed_fastq_small/*_1.trim.sub.fastq
    do
    echo "working with file $fq1"

    base=$(basename $fq1 _1.trim.sub.fastq)
    echo "base name is $base"

    fq1=~/dc_workshop/data/trimmed_fastq_small/${base}_1.trim.sub.fastq
    fq2=~/dc_workshop/data/trimmed_fastq_small/${base}_2.trim.sub.fastq
    sam=~/dc_workshop/results/sam/${base}.aligned.sam
    bam=~/dc_workshop/results/bam/${base}.aligned.bam
    sorted_bam=~/dc_workshop/results/bam/${base}.aligned.sorted.bam
    raw_bcf=~/dc_workshop/results/bcf/${base}_raw.bcf
    variants=~/dc_workshop/results/bcf/${base}_variants.vcf
    final_variants=~/dc_workshop/results/vcf/${base}_final_variants.vcf

    bwa mem $genome $fq1 $fq2 > $sam
    samtools view -S -b $sam > $bam
    samtools sort -o $sorted_bam $bam
    samtools index $sorted_bam
    bcftools mpileup -O b -o $raw_bcf -f $genome $sorted_bam
    bcftools call --ploidy 1 -m -v -o $variants $raw_bcf
    vcfutils.pl varFilter $variants > $final_variants

    done
```

Now, we'll go through each line in the script before running it.

First, notice that we change our working directory so that we can create new results subdirectories in the right location.

Output

```
cd ~/dc_workshop/results
```

Next we tell our script where to find the reference genome by assigning the `genome` variable to the path to our reference genome:

Output

```
genome=~/dc_workshop/data/ref_genome/ecoli_rel606.fasta
```

Next we index our reference genome for BWA:

Output

```
bwa index $genome
```

And create the directory structure to store our results in:

Output

```
mkdir -p sam bam bcf vcf
```

Then, we use a loop to run the variant calling workflow on each of our FASTQ files. The full list of commands within the loop will be executed once for each of the FASTQ files in the `data/trimmed_fastq_small/` directory. We will include a few `echo` statements to give us status updates on our progress.

The first thing we do is assign the name of the FASTQ file we're currently working with to a variable called `fq1` and tell the script to `echo` the filename back to us so we can check which file we're on.

Bash

```
for fq1 in ~/dc_workshop/data/trimmed_fastq_small/*_1.trim.sub.fastq
    do
    echo "working with file $fq1"
```

We then extract the base name of the file (excluding the path and `.fastq` extension) and assign it to a new variable called `base`.

Bash

```
    base=$(basename $fq1 _1.trim.sub.fastq)
    echo "base name is $base"
```

We can use the `base` variable to access both the `base_1.fastq` and `base_2.fastq` input files, and create variables to store the names of our output files. This makes the script easier to read because we don't need to type out the full name of each of the files: instead, we use the `base` variable, but add a different extension (e.g. `.sam`, `.bam`) for each file produced by our workflow.

Bash

```
    #input fastq files
    fq1=~/dc_workshop/data/trimmed_fastq_small/${base}_1.trim.sub.fastq
    fq2=~/dc_workshop/data/trimmed_fastq_small/${base}_2.trim.sub.fastq

    # output files
    sam=~/dc_workshop/results/sam/${base}.aligned.sam
    bam=~/dc_workshop/results/bam/${base}.aligned.bam
    sorted_bam=~/dc_workshop/results/bam/${base}.aligned.sorted.bam
    raw_bcf=~/dc_workshop/results/bcf/${base}_raw.bcf
    variants=~/dc_workshop/results/bcf/${base}_variants.vcf
    final_variants=~/dc_workshop/results/vcf/${base}_final_variants.vcf
```

And finally, the actual workflow steps:

1) align the reads to the reference genome and output a `.sam` file:

Output

```
    bwa mem $genome $fq1 $fq2 > $sam
```

2) convert the SAM file to BAM format:

Output

```
    samtools view -S -b $sam > $bam
```

3) sort the BAM file:

Output

```
    samtools sort -o $sorted_bam $bam
```

4) index the BAM file for display purposes:

Output

```
    samtools index $sorted_bam
```

5) calculate the read coverage of positions in the genome:

Output

```
    bcftools mpileup -O b -o $raw_bcf -f $genome $sorted_bam
```

6) call SNPs with bcftools:

Output

```
    bcftools call --ploidy 1 -m -v -o $variants $raw_bcf
```

7) filter and report the SNP variants in variant calling format (VCF):

Output

```
    vcfutils.pl varFilter $variants  > $final_variants
```

Exercise

It’s a good idea to add comments to your code so that you (or a collaborator) can make sense of what you did later. Look through your existing script. Discuss with a neighbor where you should add comments. Add comments (anything following a `#` character will be interpreted as a comment, bash will not try to run these comments as code).

Now we can run our script:

Bash

```
$ bash run_variant_calling.sh
```

Exercise

The samples we just performed variant calling on are part of the long-term evolution experiment introduced at the beginning of our variant calling workflow. From the metadata table, we know that SRR2589044 was from generation 5000, SRR2584863 was from generation 15000, and SRR2584866 was from generation 50000. How did the number of mutations per sample change over time? Examine the metadata table. What is one reason the number of mutations may have changed the way they did?

Hint: You can find a copy of the output files for the subsampled trimmed FASTQ file variant calling in the `~/.solutions/wrangling-solutions/variant_calling_auto/` directory.

Solution

Bash

```
$ for infile in ~/dc_workshop/results/vcf/*_final_variants.vcf
> do
>     echo ${infile}
>     grep -v "#" ${infile} | wc -l
> done
```

For SRR2589044 from generation 5000 there were 10 mutations, for SRR2584863 from generation 15000 there were 25 mutations, and SRR2584866 from generation 766 mutations. In the last generation, a hypermutable phenotype had evolved, causing this strain to have more mutations.

Bonus Exercise

If you have time after completing the previous exercise, use `run_variant_calling.sh` to run the variant calling pipeline on the full-sized trimmed FASTQ files. You should have a copy of these already in `~/dc_workshop/data/trimmed_fastq`, but if you don't, there is a copy in `~/.solutions/wrangling-solutions/trimmed_fastq`. Does the number of variants change per sample?

Key Points

- We can combine multiple commands into a shell script to automate a workflow.
- Use `echo` statements within your scripts to get an automated progress update.

(../04-variant_calling/index.html)

(../)