R_Pipeline_Multijob_Submission

Simon Lizarazo

2025-01-30

How can I submit Multiple Jobs in a HPC systems using R

HPC systems offer several benefits, specially if you are interested in performing repetitive tasks for large scale data. For example, it is very straightforward to create a Single Shell Script to analyze and process a single NGS experiment (From Fastq files to txt files with count matrices or peaks matrices or something else ...), but how can we approach the simultaneous analysis of many experiments?.

The way I have been doing this (There are many other ways of doing it) relies in building an integrated code in R that allows me to create multiple files that will be executed eventually.

Let's walk through a very simple experiment with artificial data. The following data frame has the following attributes for 200 experiments

- File - The name of the file I want to process
- File_Attribute - A specific attribute to my file, let's say a condition for my experiment or something related. In this example, we have 4 different conditions, Control, Treatment A, Treatment B, Treatment C

```
data = data.frame(File = paste0('SRXE', sample(c(1:1000), 200, F )),
                  File_Attribute = paste0(rep(c('Control', 'Treatment_A', 'Treatment_B', 'Treatment_C')

head(data)
```

```
##      File  File_Attribute
## 1 SRXE591      Control_93
## 2 SRXE879  Treatment_A_66
## 3 SRXE784  Treatment_B_47
## 4 SRXE853 Treatment_C_172
## 5 SRXE499     Control_197
## 6 SRXE330  Treatment_A_30
```

The goal of this exercise is to create 200 shell scripts useful for extracting counts from those .fastq files.

A standard pipeline for analyzing genomic data involves:

1. **Quality Control:**

```
The goal of this step is to Assess the quality of the raw sequences, with this in mind you identify issu
```

2. **Read Trimming and Filtering:**

   As the name entails it involves Trimming the adapters present in the sequences and remove low-quality bases according to a threshold, this improves quality data.

3. **Alignment/Mapping:**

   Map the reads to a reference genome or transcriptome. This step aligns sequences to known genomic coordinates.

4. **Post-Alignment Processing**

   Convert aligment outputs to optimize data storage and accessibility for further analysis. From here multiple analysis can be made, such as peak calling, count processing, variant analysis and more!

A key step in building a pipeline requires understanding of the algorithms used for each step. There are several different tools that can be used to analyze your data, some of them can even perform more than one task, everything depends on what you are looking for.

For this example we will use the following algorithms:

1. Trim Galore - Quality control and adaptors trimmming
2. Bow Tie - Sequence alignment
3. SAM Tools - Sorting and Indexing of SAM, BAM files
4. BED Tools - Extracting counts
5. Deep Tools - Create bigwig files.

We are going to assume that we are in our HPC environment. We need to know/create the following directories:

1. Fastqc_files : `~/experiment/fastqc/`
2. Trimmed_files : `~/experiment/trimmed/`
3. Aligned_files : `~/experiment/aligned/`
4. Count_files : `~/experiment/counts/`
5. BW_files : `~/experiment/bws/`

   Adjust this pathways accordingly!

Now, we are ready to create the shell script to submit as a SLURM job. Keep in mind that we will create 200 scripts simultaneously. We will store all these scripts in their respective folder

1. Shell files: `~/experiment/shell/`

Let's create an object for each directory.

```
fastq_directory = "~/experiment/fastqc/"

trim_file = "~/experiment/trim/"

align_file = "~/experiment/align/"

count_file = "~/experiment/count/"

bw_file = "~/experiment/bw/"

sh_file = "~/experiment/shell/"
```

Moreover we need to know the location of our reference genome and in this example a dataframe with coordinates of interest in .bed format

```
index = "~/bowtie_index/GRCh38_noalt_as/GRCh38_noalt_as"
bed_file = '~/coordinates_interest.bed'
```

With all those objects, the following steps are as simple as creating as a for loop for every file we want to analyze

NOTE: This example assumes single end sequencing. For pair end things change, but just a little in the way your code is formated.

Let's create a name reference for each experiment

```
data$nam = paste0('Name_', data$File_Attribute)
```

Now let's make our loop.

NOTE: The commands I put here are the ones I generally use. To learn more about them you can look for the documentation for each algorithm

```
for(i in 1:nrow(data)){

  run = data$File[i] #The fastq file which should be already downloaded in the fastq directory
  nam = data$nam[i] #The name we want our files to have


  bash_text =  c(
    "#!/bin/bash",
    "# ----------------SLURM Parameters----------------",
    "#SBATCH --mem=200g", ## Alocated memory
    "#SBATCH -N 1",
    paste('#SBATCH -J dataset', i, sep ='') , ## Job name
    paste('#SBATCH -o output_', nam, '.txt', sep = ''), ## Output file
    paste('#SBATCH -e error_', nam, '.txt', sep = ''), ## Error file
    paste('#SBATCH -D ', sh_file, sep = ''), ## The directory where all the outputs of your run will ap
    "#SBATCH --mail-user=enter@mail.com", ## An email in case you want to receive updates about your ru
    "# ----------------Load Modules--------------------", ## Loading the modules previously mentioned
    "module purge",
    "module load Trim_Galore/0.6.5-IGB-gcc-4.9.4",
    "module load deepTools/3.2.1-IGB-gcc-4.9.4-Python-3.6.1",
    "module load Bowtie2/2.3.5.1-IGB-gcc-4.9.4",
    "module load SAMtools/1.9-IGB-gcc-4.9.4",
    "module load BEDTools/2.26.0-IGB-gcc-4.9.4"
  )



  ## Trim galore - Trimming and quality control. A key thing here is to know how the algorithm names th
  command1 =  paste0('trim_galore --dont_gzip -o ', trim_file,
                     ' --single --fastqc ',
                     fastq_directory, run, '.fastq.gz ', sep = '')

  ## Align
  command2 = paste0('bowtie2 -x ', index,  ' -1 ',
                    trim_file, run, '_1.fq -S ',
                    align_file, nam, '.sam' , sep = '')

  ## sam to bam
  command3 = paste0('samtools view -S -b ',
                    align_file, nam, '.sam > ',
                    align_file, nam, '.bam', sep = '')

  command4 = paste0('samtools sort ',
                    align_file, nam, '.bam -o ',
                    align_file, nam, '.sorted.bam', sep = '')

  command5 = paste0('samtools index ',
```

```
                    align_file, nam, '.sorted.bam', sep = '')


  ## Extracting counts
  command6 = paste0('bedtools coverage -a ',
                    bed_file, ' -b ',
                    align_file, nam, '.sorted.bam > ',
                    count_file, nam, '.bed -counts', sep = '')


  command7 = paste0('bamCoverage --bam ',
                    align_file, nam, '.sorted.bam -o ',
                    bw_file, nam, '.bw --binSize 20 --smoothLength 60 --normalizeUsing RPGC
                    --effectiveGenomeSize 2913022398')


  full_script = c(bash_text,
                  'echo "I am starting to work"',
                  command1,
                  'echo "I just finished command1"',
                  command2,
                  'echo "I just finished command2"',
                  command3,
                  'echo "I just finished command3"',
                  command4,
                  'echo "I just finished command4"',
                  command5,
                  'echo "I just finished command5"',
                  command6,
                  'echo "I just finished command6"',
                  command7)

  sh_file_name = paste0(sh_file, 'proc_', nam, '_.pipeline.sh', sep = '')

 # writeLines(full_script, sh_file_name)

  #system(paste('sbatch ', sh_file_name, sep = ''))
}
```

Once you run this loop, for each file you will have something that look like this

```
cat(full_script, sep = "\n")
```

```
## #!/bin/bash
## # ----------------SLURM Parameters----------------
## #SBATCH --mem=200g
## #SBATCH -N 1
## #SBATCH -J dataset200
## #SBATCH -o output_Name_Treatment_C_46.txt
## #SBATCH -e error_Name_Treatment_C_46.txt
## #SBATCH -D ~/experiment/shell/
## #SBATCH --mail-user=enter@mail.com
## # ----------------Load Modules--------------------
## module purge
```

```
## module load Trim_Galore/0.6.5-IGB-gcc-4.9.4
## module load deepTools/3.2.1-IGB-gcc-4.9.4-Python-3.6.1
## module load Bowtie2/2.3.5.1-IGB-gcc-4.9.4
## module load SAMtools/1.9-IGB-gcc-4.9.4
## module load BEDTools/2.26.0-IGB-gcc-4.9.4
## echo "I am starting to work"
## trim_galore --dont_gzip -o ~/experiment/trim/ --single --fastqc ~/experiment/fastqc/SRXE962.fastq.gz
## echo "I just finished command1"
## bowtie2 -x ~/bowtie_index/GRCh38_noalt_as/GRCh38_noalt_as -1 ~/experiment/trim/SRXE962_1.fq -S ~/exp
## echo "I just finished command2"
## samtools view -S -b ~/experiment/align/Name_Treatment_C_46.sam > ~/experiment/align/Name_Treatment_C
## echo "I just finished command3"
## samtools sort ~/experiment/align/Name_Treatment_C_46.bam -o ~/experiment/align/Name_Treatment_C_46.s
## echo "I just finished command4"
## samtools index ~/experiment/align/Name_Treatment_C_46.sorted.bam
## echo "I just finished command5"
## bedtools coverage -a ~/coordinates_interest.bed -b ~/experiment/align/Name_Treatment_C_46.sorted.bam
## echo "I just finished command6"
## bamCoverage --bam ~/experiment/align/Name_Treatment_C_46.sorted.bam -o ~/experiment/bw/Name_Treatmen
##                      --effectiveGenomeSize 2913022398
```

Now you have all your files and you just need to submit your jobs! You were able to make 200 files immediately!